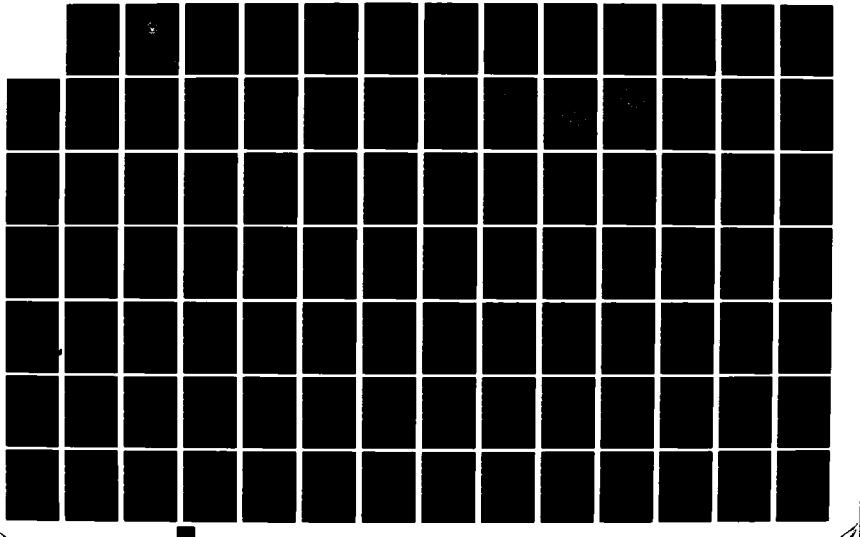
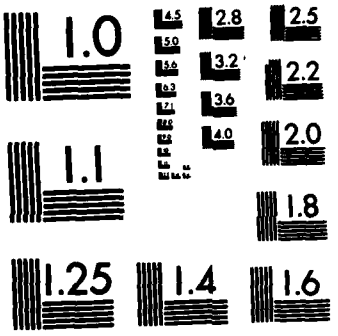
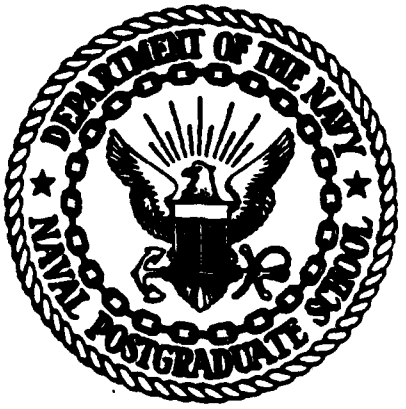
AD-A124 630 AN INTERACTIVE MICROCOMPUTER WARGAME FOR AN AIR BATTLE (U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA J O WILSON OCT 82 1/2

UNCLASSIFIED F/G 15/7 NL





NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

An Interactive Microcomputer Wargame for an Air Battle

by

James Owen Wilson

October 1982

Thesis Advisor: A. F. Andrus

Approved for public release: distribution unlimited

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A124630	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An Interactive Microcomputer Wargame for an Air Battle		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis October 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James Owen Wilson		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE October 1982
		13. NUMBER OF PAGES 136
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Wargame
Microcomputer
Interactive Wargame
APPLE III

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis is an interactive wargame using an APPLE III microcomputer (128K configuration) programmed in UCSD PASCAL. It is designed as a naval task force undergoing an air attack and is modeled from the Air Battle Analyzer by M. C. Waddell of the Johns Hopkins University Applied Physics Laboratory.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Approved for public release: distribution unlimited

An Interactive Microcomputer Wargame for an Air Battle

by

James Owen Wilson
Lieutenant, United States Navy
B.A., University of Texas, 1974

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
October 1982

Author: James Owen Wilson

Approved by: Alan R. Andrews
Thesis Advisor

James D. Esary
Second Reader

Chairman, Department of Operations Research

W. M. Woods
Dean of Information and Policy Sciences

ABSTRACT

This thesis is an interactive wargame using an APPLE III microcomputer (128K configuration) programmed in UCSD PASCAL. It is designed as a naval task force undergoing an air attack and is modeled from the Air Battle Analyzer by M. C. Waddell of the Johns Hopkins University Applied Physics Laboratory.

TABLE OF CONTENTS

I. INTRODUCTION AND BACKGROUND 7

A. AN INTERACTIVE COMPUTER ASSISTED WARGAME 7

B. WHY A MICROCOMPUTER? 8

C. PURPOSE 10

II. AIR BATTLE ANALYZER 11

A. PURPOSE 11

B. DESCRIPTION, TOOLS 11

C. LAYOUT, GAMEFLOW 15

III. USERS GUIDE 16

A. STARTUP, BUILDING THE DATABASE 16

B. PLAYING THE GAME 19

IV. AIR BATTLE COMPUTER MODEL 26

A. INTRODUCTION 26

B. GENERAL PROGRAM OPERATION 26

1. Assumptions of the Program 26

2. Program Operation, Primary Chain 29

3. Individual Programs, Briefly 31

C. THE LANGUAGE, PASCAL 32

D. FILE DESCRIPTIONS, DISK ORGANIZATION, EXECUTION TIME 33

1. Boot Disk, ABA.1 33

2. Second Disk, ABA.2 34

3. Playing Time 35

E. ORGANIZATION AND CREATION OF THE DATABASE 36
1. Filemaker and the Default Database 36
2. Organization and Format 39
3. Startem, Thesis1 39
4. Changem, Thesis2 42
V. ABAGAME, THESIS3 45
A. INTRODUCTION 45
B. MAIN PROGRAM 45
C. INITIALIZE, DOAIRLISTS, SHOWFORMS 48
D. GENERATING RADAR CONTACTS AND KILLS 49
E. DISPLAGAME AND GRAPHICS DISPLAYS 53
F. STATUS REPORTS, NEXTEVENTS, NEXTSTEP 57
G. GETKILLS, FLTUPDATE, ATKUPDATE 58
VI. FUTURE DEVELOPMENTS, APPLICATIONS 61
A. EXPANSION AND EXTENSIONS 61
B. APPLICATIONS 63
PROGRAM LISTING 65
LIST OF REFERENCES 134
BIBLIOGRAPHY 135
INITIAL DISTRIBUTION LIST 136

LIST OF FIGURES

1.1	Gaming Classification Matrix	8
2.1	Plotting Chart (Reduced)	12
2.2	Mach Meter Template (Reduced)	13
2.3	Radar Range Nomograph (Reduced)	14
3.1	Menu from Intro	17
3.2	Menu from Startem	18
3.3	Figures Used in Graphics	20
3.4	Initial Fleet Display	21
3.5	Later Fleet Display	22
3.6	Status Reports Menu	23
3.7	Nextevents Menu	24
4.1	Program Chain	30
4.2	Ship Record	37
4.3	Aircraft Record	38
4.4	Startem Flowchart	40
4.5	Changem Flowchart	41
5.1	Abagame Main Program Flowchart	46
5.2	DISPLAGAME Flowchart	54
5.3	Display Records	55
5.4	Abagame Program Constants	60

I. INTRODUCTION AND BACKGROUND

A. AN INTERACTIVE COMPUTER ASSISTED WARGAME

A wargame can be considered to be "any type of analysis or modeling that contains an explicit representation of two or more sides in defining an adversary or conflict situation." [Ref. 1] Military science has long been interested in games/gaming for their use in establishing tactical and strategic doctrine and the development of new and improved weapon systems. Low [Ref. 1] has proposed that "a morphological matrix can be constructed in three dimensions to review the field of military games in all its forms." Figure (1.1) is this classification matrix. The dimensions of the matrix are technique, scope, and application. From this matrix it is obvious that there are many possible combinations of technique, scope and application in the development of different wargames. "Clearly, any research to be performed in gaming must be selective and focused on particular elements of this matrix if the effort is not to become untenable in its proportions." [Ref. 1]

The air battle computer program of this thesis is an interactive wargame using an APPLE III microcomputer programmed in UCSD PASCAL. It is designed as a many on many engagement, to provide a tool for operations planning and evaluation, as an interactive computer wargame, and would be located as such in Figure (1.1). This wargame also has training and educational applications and therefore could be easily integrated into a wargaming or simulation course as a teaching aid due to its flexibility, ease of operation and portability.

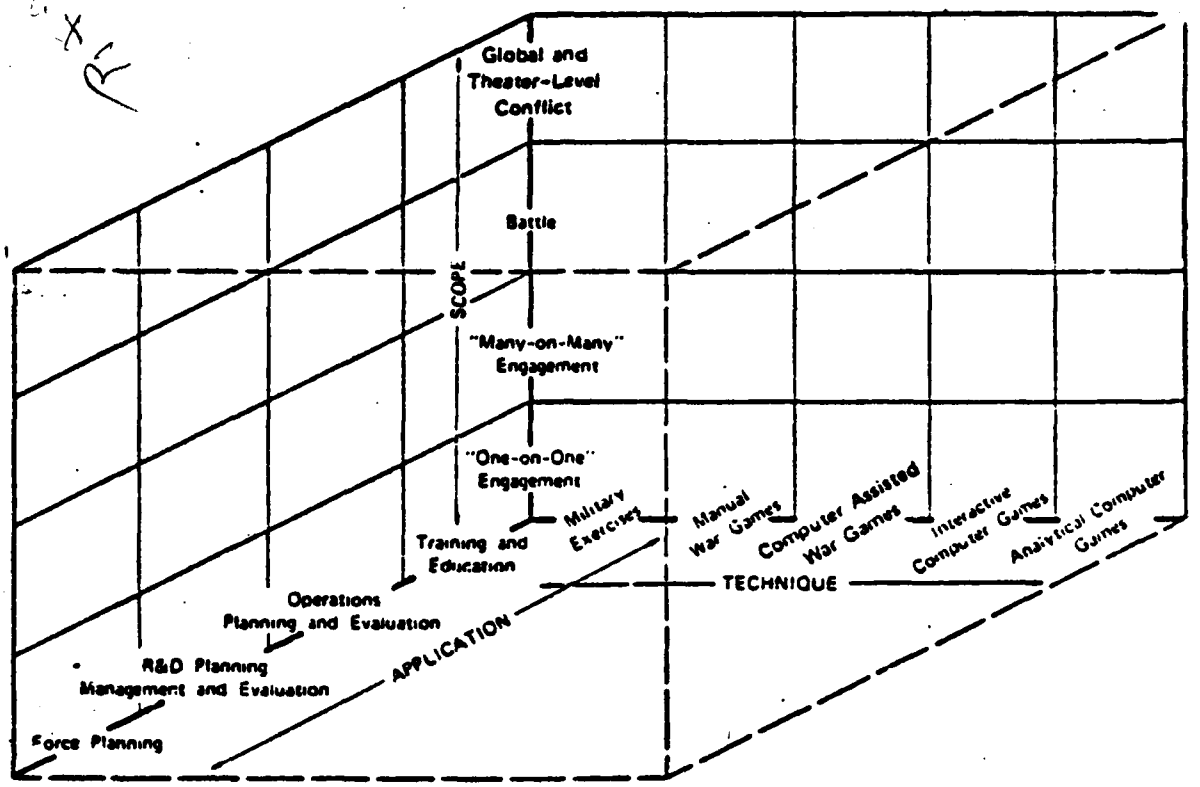


Figure 1.1 Gaming Classification Matrix

The selection of the Air Battle Analyzer as a model for this thesis was made because it provided a convenient format to initiate an interactive wargame on a microcomputer. It is designed to be general in approach and provide much flexibility in the play of the game. It also provided a very convenient and mathematical approach that was easily translated into a computer program.

B. WHY A MICROCOMPUTER?

In the last few decades, computers and the problems to which they have been applied have been a primary concern of both civilian and military communities. Computers have evolved from the UNIVAC1, vintage

1950 vacuum tube, through the transistors of the late fifties to the silicon chips and integrated circuits of today. New applications for computers are appearing everyday. Therefore, it is not surprising that the military establishment has diligently researched the applicability of the computer to its multitude of problems, e.g., guidance mechanisms for weapons, inventory maintenance of logistic support equipment, and computer wargames.

This evolution of the computer is phenomenal. The computers of today are faster, more efficient, have a larger memory capacity and, all this notwithstanding, are cheaper to build and operate. "With every major advance in solid state electronics technology, you get two new products; a smaller version of yesterday's computer and a more powerful version of today's computer." [Ref. 2] Microcomputers today are relatively cheap and are becoming a very common appliance. They can be found on board the ships and aircraft of today's Navy. They are relatively unimposing machines and most are very simple to operate. With the advent of these machines on board our ships, it has become imperative that they be used constructively.

Wargaming in the past has been conducted manually or at great expense on large computer facilities. Microcomputers offer a very pleasing alternative to the plotting and frustrating tasks of manual wargaming and provide a very economical alternative to the large computers. The NAVAL TACTICAL GAME, NAVTAG, TRAINING SYSTEM is an example of a valuable aid to teaching tactical doctrine through the use of an interactive microcomputer wargame. The NAVTAG system, however, consists of three microcomputers, three video display terminals, three mass storage devices and a printer. This system is designed specifically for NAVTAG.

C. PURPOSE

The purpose of this thesis is twofold. First, to create a prototype microcomputer version of the Air Battle Analyzer and second, to explore the capability of a standard APPLE III microcomputer.

II. AIR BATTLE ANALYZER

A. PURPOSE

The Air Battle Analyzer is published by The Johns Hopkins University Applied Physics Laboratory. It was developed in 1963 as a "means for considering in an orderly and economical fashion, how a hypothetical air battle may progress." It is to be used "in ascertaining the effects of both deployment and tactical employment of various weapon systems and other equipment on many different battle situations." [Ref. 3]

B. DESCRIPTION, TOOLS

The Air Battle Analyzer is used as a tool to bring into perspective the interactions of forces in an air battle. It is designed to provide a chronological display of movements and operations of the different units involved. The primary tool toward this goal is the plotting and display chart, Figure (2.1). This figure has several of the lines and units plotted, reflecting what the chart looks like after a battle scenario has been enacted. The chart is sectioned into three areas, a range-altitude plot, a range-azimuth plot, and a range-time plot. The link between the areas is the range scale, which runs along the horizontal axis. This common link is designed to facilitate reference from one plot to another.

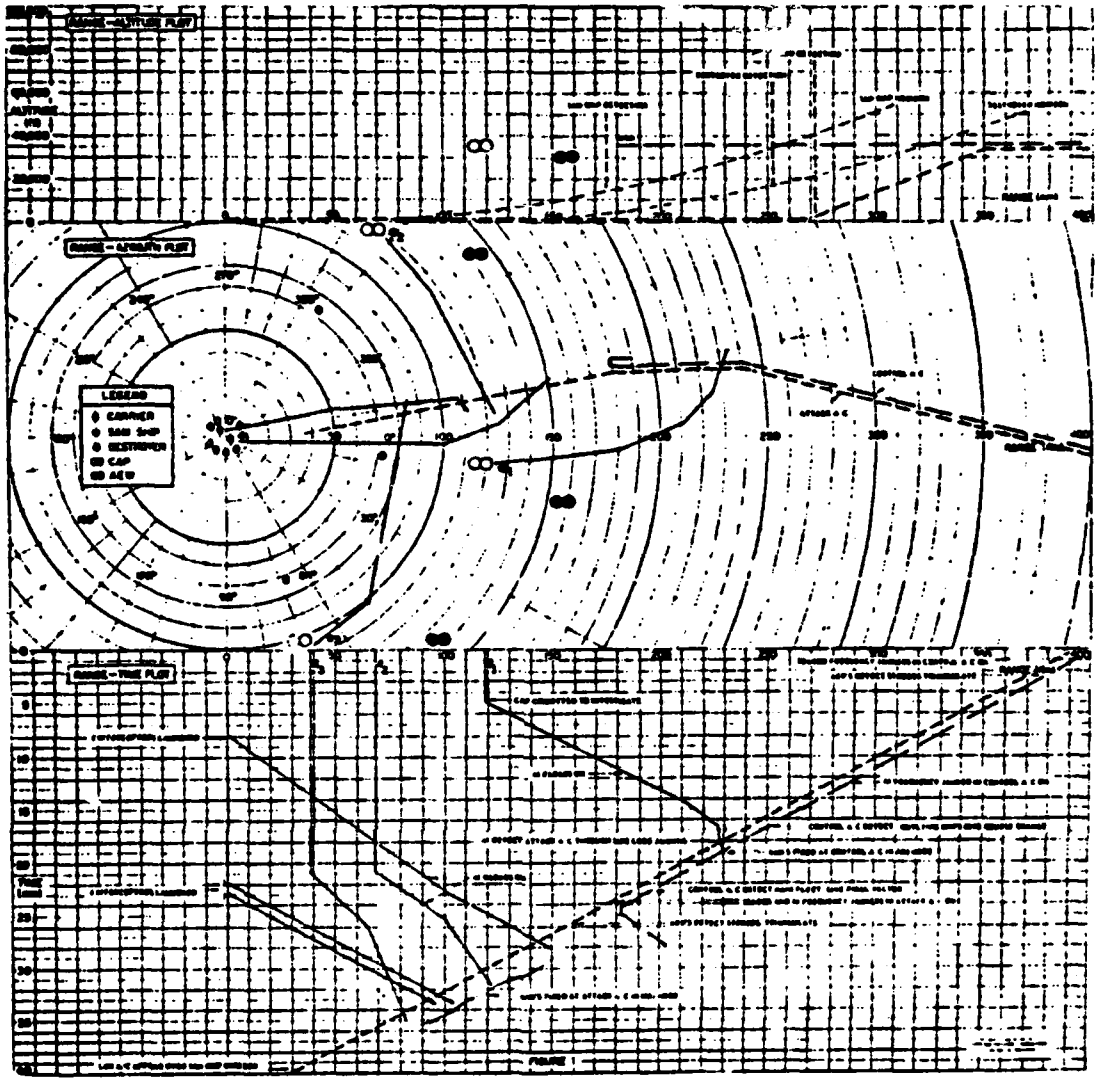


Figure 2.1 Plotting Chart (Reduced)

Included with the chart are several nomographs and plastic plotting tools that are essential for several of the numerical calculations involved, and that facilitate the actual plotting of lines on the different areas of the chart. Figures (2.2) and (2.3) are examples of the associated tools for use with the Air Battle Analyzer. Figure (2.2) is

a Mach Meter, used "for drawing aircraft range-time lines having slopes appropriate to the aircraft speed." Figure (2.3) is a nomograph for determining radar detection ranges in a clear environment.

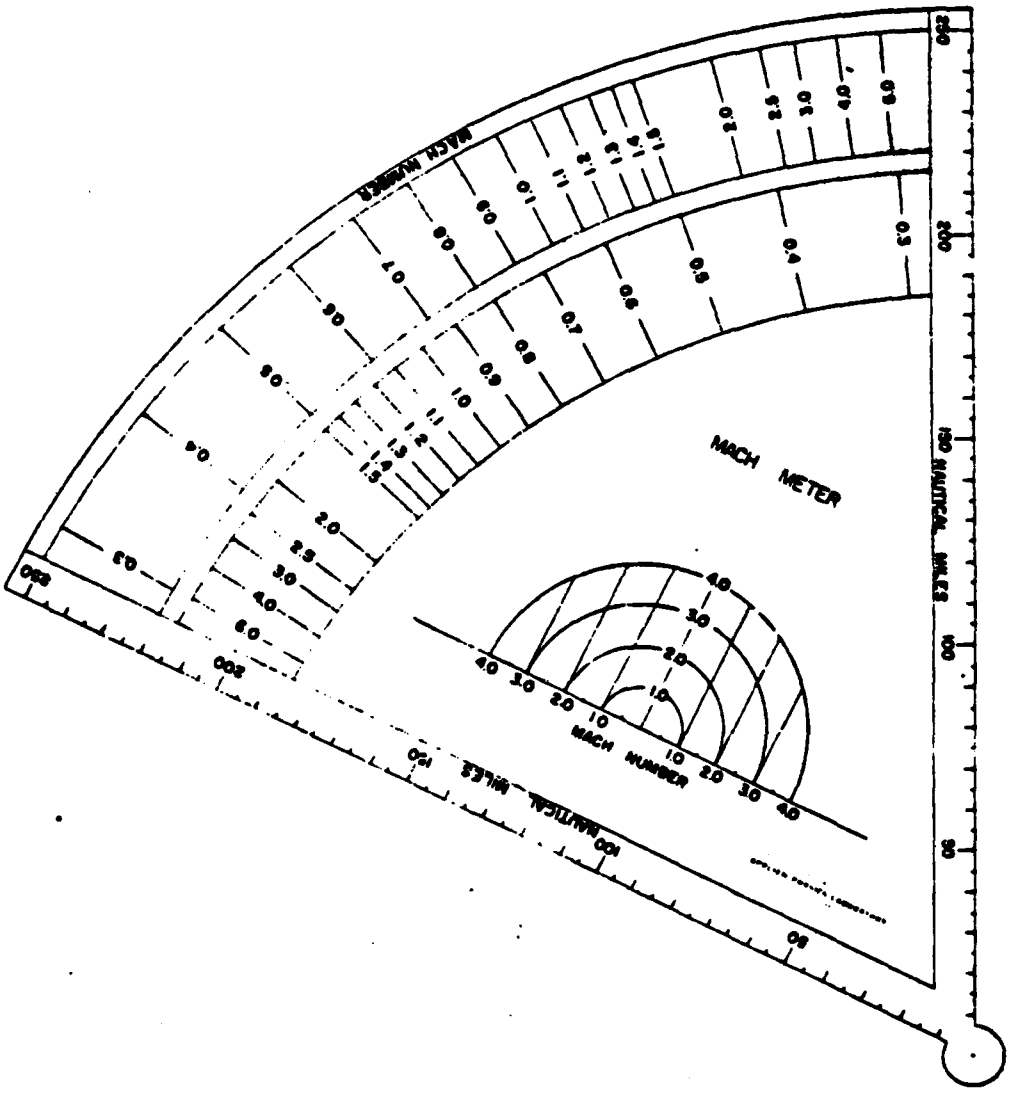


Figure 2.2 Mach Meter Template (Reduced)

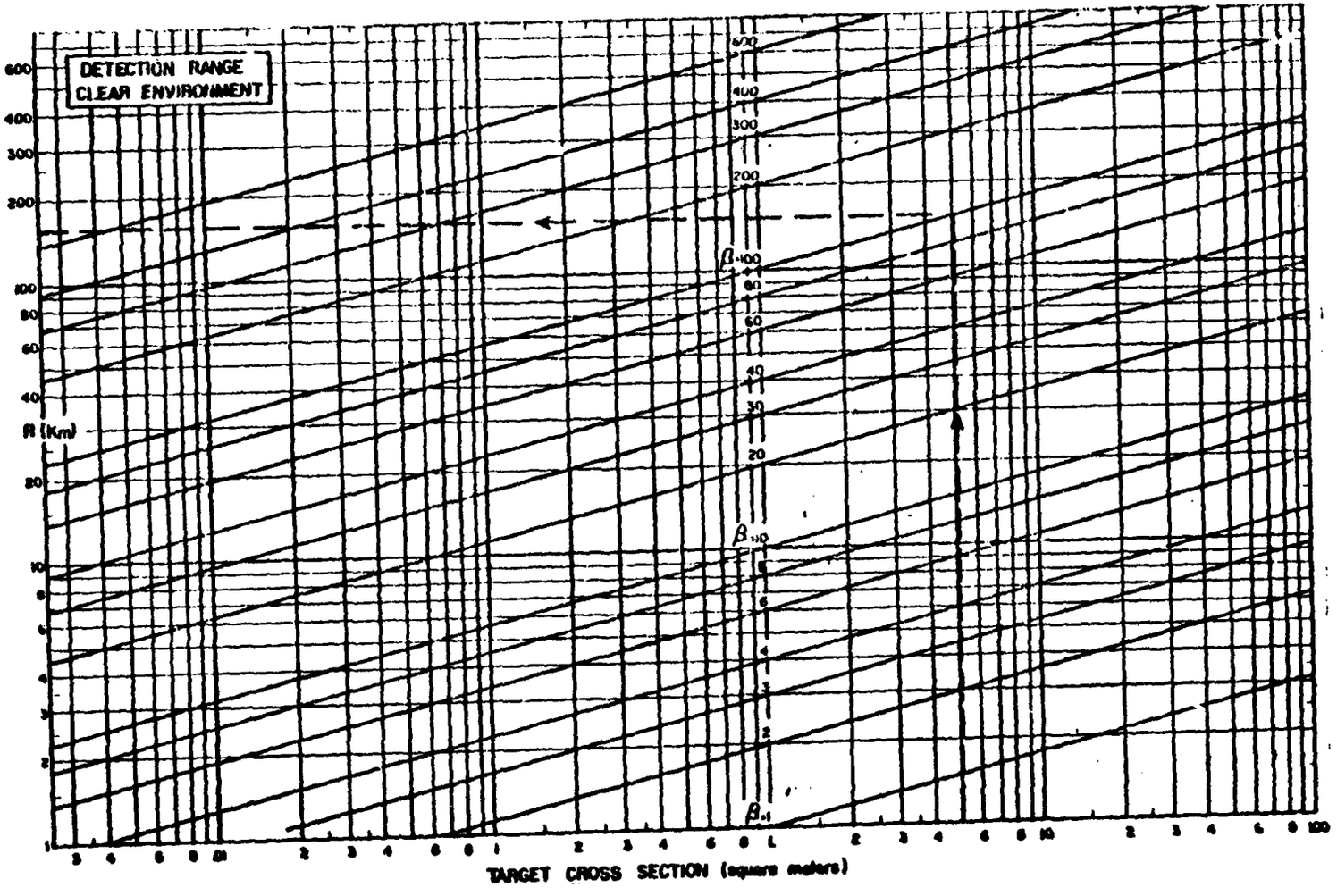


Figure 2.3 Radar Range Nomograph (Reduced)

C. LAYOUT, GAMEFLOW

To begin the analysis, all the units involved and the radar horizons of the fleet units are plotted on the range-azimuth area of the chart. On the range-time area are plotted the early parts of the attack profile, combat air patrol and early warning aircraft. Also needed is data concerning the performance characteristics of the units and the weapons of both the fleet and attack. Additionally, battle plans for both the fleet and attack are needed in order to stage the actual attack and subsequent defense. These battle plans and performance characteristics are left entirely to the user, but should appropriately reflect what the user desires to analyze. "The battle plans may be thought to consist of the orders of battle and of standard doctrine and such information as might be given in a pre-attack briefing." [Ref. 3]

The game progresses as the interactions between the different units begin to unfold and information is "received" concerning the attack. These interactions are indicated by the attack profiles on the range-altitude chart intersecting with the radars of the fleet units. These interactions enable the user to make decisions concerning the use and deployment of his forces, provided the appropriate employment of the battle plans.

The examination of the battle is completed when the user decides the fleet has become fully aware of the nature of the attack and has brought to bear any and all of the units he believes should be used in its defense. At this point, the analysis and "second guessing" can be done to try and determine how the battle might have progressed had different decisions been made at certain points in the progression of events.

III. USERS GUIDE

A. STARTUP, BUILDING THE DATABASE

The Air Battle Computer model is designed to operate on an APPLE III microcomputer with one additional disk drive and a video monitor. The needed software is stored on two 5 inch floppy disks, labeled ABA.1 and ABA.2. The game will start automatically following these simple instructions:

i) Place disk labeled ABA.1, label up, in disk drive 1 (built in drive).

ii) Place disk labeled ABA.2, label up, in disk drive 2 (external drive).

iii) Turn video monitor on.

iv) Turn APPLE III computer on.

The disk drives will whir audibly, and shortly thereafter the APPLE III PASCAL copywrite notice will appear briefly on the monitor, followed by two screens of very general information about certain aspects of program operation. It is important to note the orientation of the screen, and that the figures on displays are disproportionately large relative to the distance between units.

The third screen presented will be the first selection menu, Figure (3.1). After the selection from the first menu the disk drives will whir briefly as program control is chained to the programs on the second disk. What follows will depend on the menu choice of the user. If option 1 was selected, a screen titled FLEET COMPOSITION will appear. This screen is followed by six others that present a general overview and specific parameters for the ships of the default database.

If you have played this game before, then you may be familiar with the default input database and/or you may have previously remodeled it to suit your needs.

Please select one of the following options:

1 : BUILD DATABASE; WITH REVIEW of default database.

2 : BUILD DATABASE; WITHOUT REVIEW of default database.

3 : Use the DEFAULT DATABASE parameters with NO REVIEW.

4 : Use the DATABASE parameters retained FROM LAST GAME.

Type a number from 1 to 4 :

Figure 3.1 Menu from Intro

Following the review of the fleet, the review of the aircraft will begin with a screen titled AIRCRAFT. The first screen describes how friendly aircraft are deployed, followed by two screens of specific parameters for deployed fighters and three screens of specifics on deployed early warning aircraft. These six screens are followed by one screen of specific parameters for each of the eighteen attack aircraft. Once the user becomes familiar with the format, these screens can be advanced quickly. If the user desires to change some of the parameters, these same values will be presented again during the process of changing the default database.

The selection menu of Figure (3.2) follows this review of the default database. For the novice user it is recommended that selection 1 be chosen, primarily because it does not require further manipulation with the database prior to seeing it, using it and becoming familiar with it during the game portion of the program. Thus, this selection

bypasses a possible detrimental aspect of the game for the novice user, namely, lengthy, boring and repetitious review of parameters. After the user has become familiar with the default database through use of the game he is better able to decide what aspects of it he may wish to change or enhance. After this selection there is a short period of disk activity while the default database files, located on disk number one are being transferred to the game database files on disk number two and the control is transferred to the game program.

How do you wish to set up the players?

1 : Use the default fleet/ship and aircraft database.
2 : Use the default fleet and build your own aircraft.
3 : Use the default aircraft and build your own fleet.
4 : Build your own fleet and aircraft database.

Type a number from 1 to 4 :

Figure 3.2 Menu from Startem

If the user decides to build a new database, he is given the opportunity to add units, delete units, or change unit parameters. This is done on a question and answer type basis with the program initiating the questions. The user is first given the opportunity to alter the ships' database provided the default fleet is not used. For each ship, he is asked if he wishes to delete it from the database. If he answers negatively, he is asked if he wishes to change any of its parameters. If he answers no, he is asked the same questions for the next ship, etc. If he answers positively to the deletion, he is

immediately asked the same question for the next ship. If he answers yes to changing a ship's parameters, he is shown each parameter for that ship and asked if he wishes to change it. A positive response then causes the program to ask him to enter the new value, a negative response causes the program to retain the old value. In either case, the program progresses to the next parameter value and the process is repeated. After all the ships in the default database have been presented, the user is asked if he wishes to add any ships to the database. If ships are added, he is asked to enter the appropriate parameter values.

When the fleet game database has been built, the user is given the same opportunity to add, delete, or change parameter values of the attacking aircraft. The screen and questioning format for changing the attack aircraft is the same as that for the fleet units.

B. PLAYING THE GAME

The first graphic display, Figure (3.3) is the next screen presented to the user. This display simply introduces the shapes that will be seen on the displays of the fleet and attack. The figure for the fighter aircraft is used on both the fleet display and the attack display; however, the attack display is separate and the user has the option of viewing it.

Next the user is asked if he desires the computer to step through the game at a specified time step. It is recommended that this be answered NO. The program then begins to check for any radar contacts. This requires scanning the linked lists several times, which for the first few time steps causes a rather lengthy pause in visual activity.

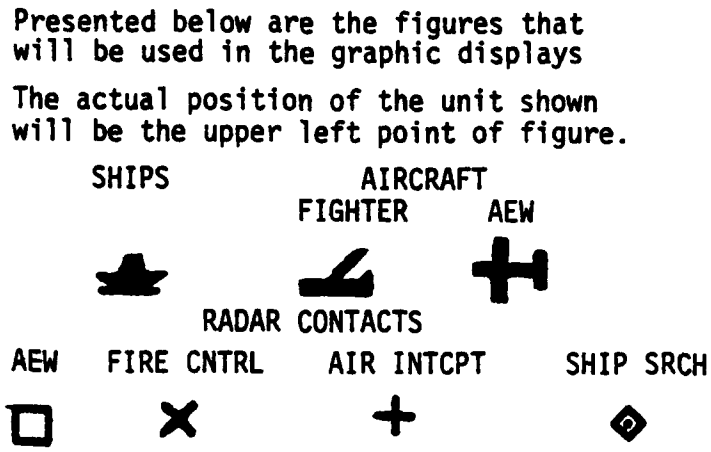


Figure 3.3 Figures Used in Graphics

During this pause the program provides a visual indication[1] that it is still operating. The first display of the fleet is then presented with the center of the screen cluttered with several units drawn on top of each other. Along the top of the screen appears:

U(pscale / D(ownscale / R(ecenter

Along the bottom of the screen are a variable length line which portrays 10 nautical miles on the screen scale, and below that:

C(ontinue Time : xxx

The time is the current game time, in minutes. By pressing "D" or "U", the user can downscale or upscale the display. A few downscales at this point will begin to display the units on a less cluttered scale. The user can recenter the display on any unit by typing "R" and the

[1]The symbol "=>" is printed diagonally across the screen from the upper left corner of the screen.

number of the unit. A unit may not appear on the screen due to the scale and the X-Y values of the screen center. The user enters "C" when he is familiar with this view of the force layout and is ready to continue the program. The user is then asked if he wishes to see the attack; if so, it is displayed in the same fashion as the fleet.

Figures (3.4) and (3.5) are examples of the screen display of the fleet. Figure (3.4) exhibits the initial display, downscaled a few times, and Figure (3.5) exhibits a later display with several radar contacts and other game consequences.

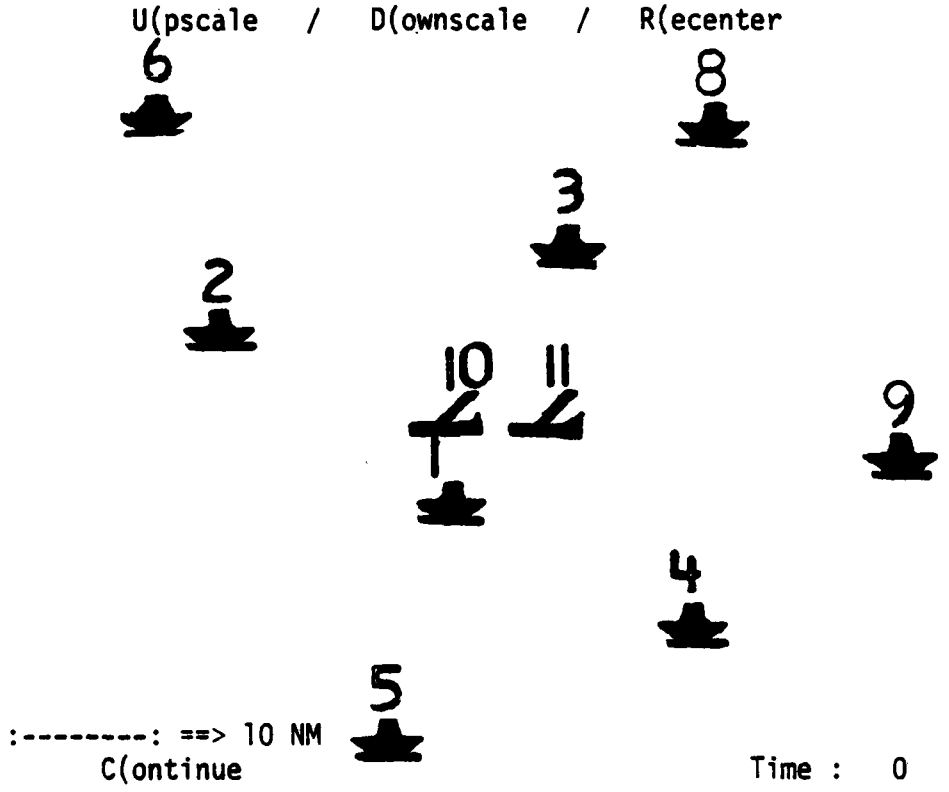


Figure 3.4 Initial Fleet Display

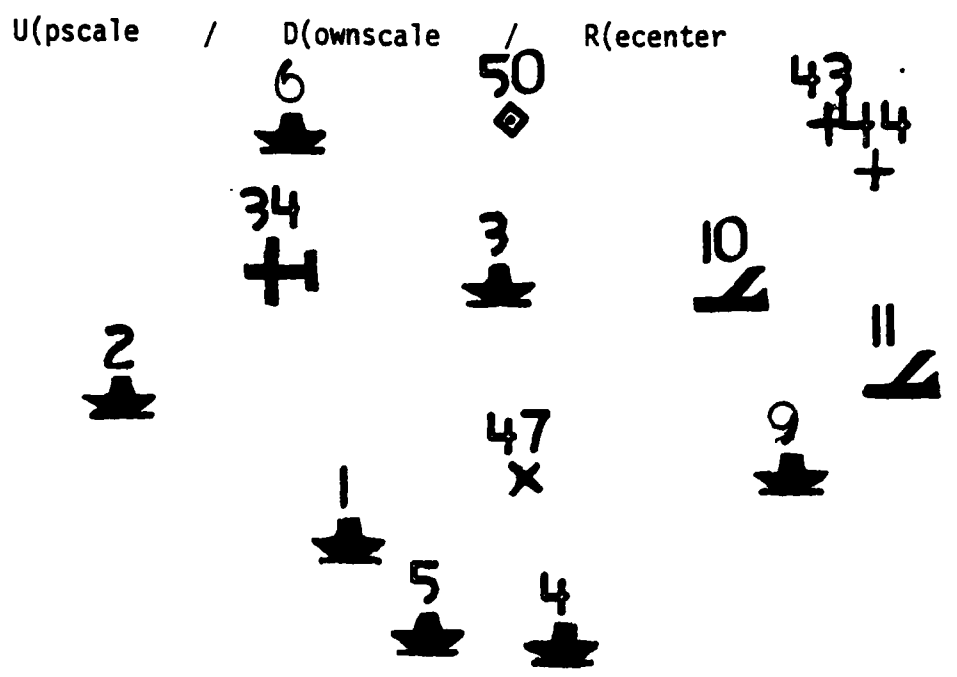


Figure 3.5 Later Fleet Display

The next presentation is the menu for selecting status reports, Figure (3.6). These reports are amplifying information about the fleet display. For example, the fighter/interceptor report contains the unit number corresponding to the number on the graphic display, and the aircraft's coordinate position, position relative to the carrier, heading/course, velocity, the remaining minutes of endurance and the number of missiles remaining. The early warning aircraft report has exactly the same layout and information as the fighter/ interceptor with the exception AEW aircraft do not carry missiles. The ship report indicates the unit number, ship type, coordinate position,

Please select an option according to which status of forces report you wish to peruse.

1 : Ships.
2 : Fighter / interceptor.
3 : Early warning aircraft.
4 : Radar contacts.

Q : Quit.

NOTE :: COORDINATE POSITIONS are SCALED : 1 = 10 NM.

After your selection you will be presented with specifics and status information concerning your selection. You will then be returned to this menu where you may make another selection or repeat a previous selection.

Figure 3.6 Status Reports Menu

course and speed, the number of long and short range missiles still onboard, and the number of missile hits and bomb hits. The radar report contains the contact number, coordinate position, the contacting radar type, contacting unit's number, and if the contact has been killed on this step.

After studying these reports, the user is presented the opportunity to redeploy or move the friendly units. The next menu, Figure (3.7), allows the user to specify how the friendly units are to move during the next time step. The selections allow the user to launch an aircraft by selecting aircraft type, and entering the desired heading, velocity and altitude. The user can initiate the recovery of an aircraft. For example, if a fighter was out of missiles, or was getting "low" in endurance, by selecting 'recover a fighter' the airborne fighters' unit numbers and positions relative to the carrier are

Please choose your desired course of action :

1 : Move a fighter/interceptor.
2 : Launch a fighter/interceptor.
3 : Recover a fighter/interceptor.
4 : Move an AEW aircraft.
5 : Launch an AEW aircraft.
6 : Recover an AEW aircraft.
7 : Move/manuever an individual ship.
8 : Alter the PIM / SOA of the fleet.

D : Review the display of forces.
R : Review the status of forces.

Q : Quit.

After your selection, you will be asked for specifics about your selection, then you will be returned to this menu where you may make another selection, repeat a selection for another aircraft/ship or stop.

Figure 3.7 Nextevents Menu

presented individually. For each aircraft, the user is asked if he wishes it recovered. He simply answers yes for the aircraft he wishes to recover; if he answers no to each fighter, then the program returns to the selection menu. When a recovery is initiated, the program directs the aircraft towards the carrier. When the "recovered" aircraft gets within five nautical miles of the carrier, an "instant landing" is performed. The user can alter the heading, velocity or altitude of an airborne aircraft or he can change the course and speed of the fleet or of an individual ship of the fleet. He can also review the force displays or status reports from this menu.

After making all desired changes, the user is shown the current game time, the default game end time, and is asked if he wishes to stop the program. If the program is not stopped, he is asked to enter the

next time step length. Time steps do not have to be equal in length or any specified minimum or maximum length. However, if a step of greater than 60 minutes is entered, the program asks the user to verify the entry. The time step must be entered in minutes. After this entry, the entire game process is then repeated and the user will see the same visual indications of program activity while the program checks for radar contacts. This is followed by the next display of the fleet. If any movements were ordered from the event menu, the units will be displayed at appropriate relative positions, and any radar contacts, if made, will also be displayed.

The program operates on a time step structure. This is important to note because all calculations are performed at the end of each time step. The time step increment is supplied by the user. No calculations are made for interactions that would have occurred between time steps. Therefore, when interactions between the forces have begun, it is recommended that time steps of no more than 5 minutes be used. Using the default database, the first radar contacts are made after approximately 30 game minutes and interactions will begin after approximately 45 game minutes.

IV. AIR BATTLE COMPUTER MODEL

"Programs that use procedures well are generally far easier to read, easier to understand, easier to change, and easier to get working." [Ref. 4, p. 90]

A. INTRODUCTION

This chapter and the next explain the program execution. This chapter is written in four sections. The first will list and explain the assumptions and give a brief overview of the program. The next states a few arguments for the programming language choice. This is followed by a description of the disks and the files residing on them. The last section explains the organization and creation of the database, including how the default database is formed and altered.

B. GENERAL PROGRAM OPERATION

1. Assumptions of the Program

This program was designed to be as similar to the Air Battle Analyzer as possible; however, there were several simplifying assumptions that were required. Some of these assumptions were necessitated by memory space limitations in the actual game portion of the program, as written for the APPLE III. The assumptions are;

i) default database used,

ii) one-sided play,

iii) radar detections and missile firings,

iv) cartesian coordinate system,

v) instantaneous manuevers,

vi) time step processing.

The following paragraphs briefly describe each assumption.

A default database has been supplied with the program. This was done to allow the novice player to use the program immediately, without a lot of foreplanning concerning characteristics and battle plans. The more experienced player can easily change this database. By altering the database, the player can see the differences these changes make in the outcome of the game.

The attack battle plan is written into the program, i.e., the attack performs only preplanned[2] manuevers. This was done in order to hasten the play of the game as well as to keep it simple for the user. This allows the user to concentrate on the fleet's battle plan without the distraction of assuring the attack follows his plan. Also, this frees the user to experiment with many battle plans against a common attack profile.

The default attack force consists of eighteen aircraft located approximately 500 NM East of the fleet. The force is formed in six wings of three aircraft. All of the attack aircraft begin the game on a heading of 270. The first three aircraft begin at 20,000 feet altitude and the rest are at 10,000 feet. The first three and the last six aircraft begin the game at 350 knots and the remaining aircraft

---

[2]These manuevers are preplanned in the sense that they occur after certain time periods. The attack "flight" profiles may change from game to game because of different attack databases or different time step lengths used throughout the game.

start at 400 knots. As the game time exceeds multiples of 20 minutes, each aircraft's heading is altered towards the carrier. When the first aircraft gets within 200 NM of the carrier the altitude of each aircraft is changed to 200 feet. When the game time exceeds 120 minutes the remaining aircraft begin their retreat at 10,000 feet, heading 090 at 450 knots.

The attack profile portion of the program is contained in the ATKUPDATE procedure which resides on the Thesis3b file. This profile is programmed in a way that should be easy to change. Most of the parameters that are necessary to create the profile are written as constants in the declarations section of the game program, residing on file Thesis3. Further detail on this procedure can be found in Chapter 5, with possibilities of expansion discussed in Chapter 6.

Radar detection of an attack aircraft occurs when it comes within a certain finite range of the friendly unit. That range is the lesser of the maximum radar range or the radar horizon, a function of the aircraft(s) altitude. In order for a unit, attack or friendly, to fire a missile or drop a bomb, the unit must have an available missile (bomb) and the target must be within minimum and maximum missile range. For air intercept radars, air-to-air missiles, and air-to-surface missiles, the target must also be within the detection/firing envelope.

The playing area is an X-Y cartesian coordinate type layout, with positioning of aircraft and ships referenced to their X-Y positions and altitudes. The X-Y axes are measured in nautical miles and altitudes are measured in feet. Altitude changes, and turns are done instantaneously rather than gradually over some time and distance.

The game portion of the program uses a time step structure vice a next event structure. This method was chosen because it allows the operator to increment the battle scenario at a pace of his own choice. Also, this structure provided a more convenient approach to the interactive aspect of the program. A combination time step, next event structure would provide a more realistic methodology; however, this project's completion date necessitated the strict time step approach. The major drawback to this approach is that all interactions between the fleet and the attack are done at each time increment. This necessarily implies that a lengthy time step, once interactions between the opposing forces have begun, might cause a missed interaction, when in fact one certainly would have occurred.

2. Program Operation, Primary Chain

The Air Battle computer program is composed of four distinct operational units that are chained together such that program flow passes from one logical stage of operation to another. The four units are depicted in Figure (4.1) in a simple flow diagram. These units are actually four distinct, separately compiled, program codefiles. The APPLE library unit CHAINSTUFF was used as a mechanism for the chaining of the programs. The chaining is accomplished by declaring, at the start of the current program, the filename of the program codefile that is to be executed when the current program concludes.

CHAINSTUFF also provides a construct that allows a string variable to be passed between the programs, thus permitting a very convenient structure to allow information obtained in one program to be referenced in the next.

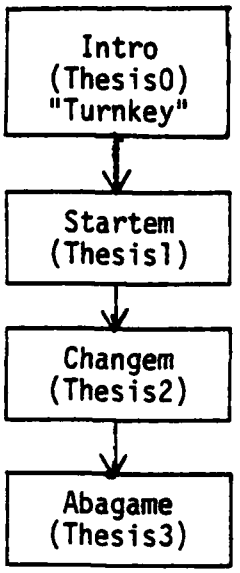


Figure 4.1 Program Chain

This mechanism is of primary importance in the sequence control of an interactive computer program, i.e., how the user directs the sequence of program operations. The "dialogue" or sequence of exchanges between the user and the program is the user's tool for this control. Ramsey and Atwood [Ref. 5] identify eight general dialogue types, of which this program uses two that require little or no training:

i) question and answer,

ii) menu selection.

These two choices, specifically the first, affect other aspects of an interactive computer program, namely the data entry. Further detail on this area of the program will be discussed in the section on building the database.

3. Individual Programs, Briefly

Intro (file Thesis0), the "turnkey" program, is a short initialization program that presents the user introductory information pertaining to the operation of the program. It also provides a notification of certain aspects of the program execution. It contains the initial option selection menu as well. From this selection, control passes to one of the other three programs.

Startem (file Thesis1) is executed unless the fourth option (use last game's database) is selected from Intro. This program presents the parameters established in the default database. After the review, the next selection menu is presented to allow the user to select which of the main areas of the default database he may wish to change or retain. The option selected from this menu determines what is done in Changem (file Thesis2), the program that builds the game database.

Changem is the program in which the user is presented the opportunity to add or delete units, or change parameters of the units in the fleet or attack. The user first is asked if he wishes to retain a unit; if he chooses to do so, he then is asked if he wishes to change any of its parameters. If he answers positively to this, he is presented with each of the parameters individually and asked if he wishes to change it; if so, he is asked to enter the new value.

Abagame (file Thesis3) is the actual game program. This program reads the game database built in Changem and forms linked lists of the records, separating the aircraft records into an "attack" list and an "air" list. It forms and displays the graphic figures, then

goes into a loop for the manipulations of the game for each time step.

C. THE LANGUAGE, PASCAL

> "The idea of separating a program into general procedural units that operate on data, and then establishing communication between the units by passing data structures back and forth is the essence of good computer programming. A language like PASCAL gives you a good opportunity to develop this strategm since PASCAL provides both procedural subunits and a large assortment of data structuring techniques." [Ref. 4, p. 274]

PASCAL was chosen as the programming language for several reasons. First, the languages the APPLE III computer supports limited the choice to either BASIC or PASCAL. Second, the expected length of the program necessitated a choice that would provide a document that could be easily read and easily changed. Third, the general nature of PASCAL provides a natural top-down structuring of a program.

Primarily PASCAL is much easier to read and understand than BASIC. This is important if a program may be changed at a future time, by either the original or some other author. PASCAL is much better suited to lengthy programs than BASIC due to the procedural subunits available and the ease in which a program is broken down into those units. PASCAL naturally eases the programmer into top-down development techniques.

There are several aspects of APPLE III PASCAL that lend themselves to the memory space problems, some positively, some negatively. One option of the APPLE III PASCAL system seemed to have very promising attributes to the memory space problem. The reference manuals indicate the system allows the program to control what library units will be

"loaded" into the computer memory by the compiler options "NOLOAD" and "RESIDENT". Without these options, all library units "used" by the program are loaded into memory at the start of execution, thus reducing memory available for the program. However, with these options, the program is supposed to be able to control when a unit is "loaded" into memory, thus a unit can be "resident" only when it is needed and not resident otherwise. The author was able to get this option working for all but the graphics unit, PGRAPH. Therefore, unfortunately, it was not included in the program.

Another APPLE III PASCAL characteristic that is similar to the "noload" option is program segmentation. A program can use 15 segmented procedures, which are loaded into memory only when they are active. This aspect was used in the Abagame program, and indeed was essential.

Still another area of UCSD PASCAL that suggested flexibility with memory space allocation was the use and reuse of memory for dynamic linked lists. However, the method the APPLE III uses is not as flexible and therefore was not as useful as was hoped. The computer does not "dispose" of deleted records on an individual basis but rather they are "disposed" in a block of consecutively linked records. This aspect was indeed useful, but it was not as flexible as one that would allow the reuse of memory space occupied by a single dynamic variable after "disposal".

D. FILE DESCRIPTIONS, DISK ORGANIZATION, EXECUTION TIME

1. Boot Disk, ABA.1

Included on the Boot Disk are all the files needed to boot the APPLE III's Sophisticated Operating System (SOS) files, sos.kernel,

sos.interp, and sos.driver as well as the PASCAL files system.pascal and system.miscinfo. These five files are needed to boot the PASCAL system. The only drivers supported in the sos.driver file are the CONSOLE and GRAPHICS drivers. Memory space limitations in the game portion of the program were the reason no other drivers were supported in the sos.driver files. These five files occupy 165 blocks on the disk. Additionally, the system.library codefile, the system.startup codefile and the default database reside on the Boot disk.

The system.library occupies 70 blocks and contains the APPLE library units, APPLESTUFF, CHAINSTUFF, LONGINTIO, PASCALIO, REALMODES, TRANSCEND, and PGRAPH. The system.library also contains the unit THESISTUFF, in which reside several procedures designed specifically for use in each of the Air Battle Analyzer programs. The type declarations needed in each of the programs are also in the Thesistuff unit. READINT and a slightly modified version of it, READREAL, were obtained in an article written by Edward Heyman [Ref. 6]. The system.startup codefile is the name given to the codefile of the Thesis0 text file.[3] This is the short introductory program with the first option selection menu. This file occupies 8 blocks on the disk. The last two files on the boot disk are the files containing the default database, together they occupy 5 blocks.

2. Second Disk, ABA.2

The files residing on the second disk are the last three of the program codefiles, Thesis1, Thesis2, Thesis3, and the Thesis3.lib(rary)

[3]This is a naming convention on the APPLE III for a "turnkey" program, a program that executes immediately after the system is booted up.

occupying 17, 25, 43, and 11 blocks respectively. As explained earlier, Thesis1 is the codefile for the STARTEM program, Thesis2 is the codefile for CHANGEM and Thesis3 the codefile for ABAGAME. Also residing on this disk are the game database files created in the CHANGEM program, and the "outfile" database files that are created in ABAGAME. These four database files will have varying lengths, depending on the changes instituted in CHANGEM and the actual play of the game.

There are three units, MAKEFORMS, GRAFSTUFF, and BEARINGS on the Thesis3.library file. MAKEFORMS forms the packed arrays that represent the forms shown in the graphic displays. GRAFSTUFF has the UPSCALE, DOWNSCALE, and RECENTER procedures used in the graphic displays. BEARINGS has the procedures that get distances between two coordinate positions, DISTANCE; bearing from one position to another, DEGREES; and a new coordinate position after a time step has occurred, GETNEWXY.

3. Playing Time

This program can be executed in two stages;

i) review and build a database,

ii) and play the game.

To do a thorough job of each entails about 30 to 45 minutes per stage. When a user builds a database, it is retained on the second disk as game database files. These files are not altered by the game program. Therefore, this game database can be used as often as desired. For each separate game, the player can simply alter his deployment of forces and/or change time step lengths to view a new battle unfold.

With each execution, of course, the user becomes more familiar with the required key strokes necessary to accomplish his goals and thus the playing time decreases. Likewise, after building a few databases for the game, the user again becomes familiar with the key strokes required and thus can accomplish the first portion in less time.

E. ORGANIZATION AND CREATION OF THE DATABASE

1. Filemaker and the Default Database

Figures (4.2) and (4.3) illustrate the record types used in the program; Figure (4.2) is the ship record and Figure (4.3) is the aircraft record. These figures illustrate the variables contained in each record, the range of each variable, and the meaning of ambiguous variable names. Note that the aircraft record is a multiple nested variant record. An aircraft can be an "enemy" or "frend" and if it is a "frend", it can be either "intcpt" (interceptor/fighter) or "aew" (early warning aircraft).

Filemaker is the program that creates the default database by initializing the variables in each record. This program defines the number and types of units involved and then assigns parameter values to each unit. If a new default database is desired, the program Filemaker can be used to form it. The user would need to edit the text version of Filemaker and change whatever aspect of the file is desired. After saving this textfile, he would need to recompile the new Filemaker text and then execute the compiled code to create the new default database files. The filenames of the default database are SHIPINFILE and AIRINFILE. The parameter values assigned to the units in the default database exhibit a close resemblance to the parametric values suggested in the manual game's example.

```
ship = record
         link  : shipntr;
         class : shiptype;
         num   : 0..255;
         xpos  : real  ;
         ypos  : real  ;
         pim   : 0..359;
         soa   : 0..50;
         fcrng : 0..255;
         ssrng : 0..255;
         lrsam : 0..255;
         srsam : 0..255;
         lrmpk : real   ;
         srmpk : real   ;
         srmin : 0..255;
         srmax : 0..255;
         lrmin : 0..255;
         lrmax : 0..255;
         mhits : 0..255;
         bhits : 0..255;
         sunk  : boolean;
       end;
```

Variable Meanings:

```
link  : A pointer variable used in the linked lists.
class : A program defined type (cv, dest, crsr).
num   : The unit number, assigned in Abagame.
xpos  : The X coordinate position of the unit.
ypos  : The Y coordinate position of the unit.
pim   : Course/Heading of the unit.
soa   : Speed of the unit.
fcrng : Fire control radar maximum range.
ssrng : Ship search radar maximum range.
lrsam : Number of long range SAM's.
srsam : Number of short range SAM's.
lrmpk : Long range missiles' probability of kill.
srpmk : Short range missiles' probability of kill.
srmin : Short range missile minimum target distance.
srmax : Short range missile maximum target distance.
lrmin : Long range missile minimum target distance.
lrmax : Long range missile maximum target distance.
mhits : Number of missile hits endured.
bhits : Number of bomb hits endured.
sunk  : True if (mhits + bhits) GT a program constant.
```

Figure 4.2 Ship Record

```
aircraft = record
             link     : airpntr;
             num      : 0..255;
             xpos     : real  ;
             ypos     : real  ;
             alt      : 0..30000;
             azmth    : 0..360;
             velcty   : 0..2000;
             iff        ifftype;
             case ifftype of
               enemy  :
                 (       asm     : 0..31;
                         asmrng  : 0..255;
                         asmenv  : 0..359;
                         asmpk   : real   ;
                         bomb    : 0..31;
                         bombpk  : real           );
               frend  :
                 (       acfrnd  : frndtype;
                  case frndtype of
                         intcpt  :
                           (       intndr : 0..255;
                                   airng  : 0..255;
                                   aienv  : 0..359;
                                   aam    : 0..31;
                                   aamrng : 0..255;
                                   aamenv : 0..359;
                                   aampk  : real   );
                         aew     :
                           (       aewndr : 0..255 ;
                                   aewrng : 0..255 ));
             end;                record
```

Variable Meanings:

link	: A pointer variable used in the linked lists.
num	: The unit number, assigned in Abagame.
xpos	: The X coordinate position of the unit.
ypos	: The Y coordinate position of the unit.
alt	: Altitude of theunit.
azmth	: Course/Heading of the unit.
velcty	: Ground speed of the unit.
iff	: A program defined type (enemy, frend).
asm	: Number of Air-to-Surface missiles.
asmrng	: Air-to-Surface missile range.
asmenv	: Air-to-Surface missile firing envelope.
asmpk	: Air-to-Surface missile probability of kill.
bomb	: Number of bombs
bombpk	: Bombs' probability of kill.
acfrnd	: A program defined type (intcpt, aew).
intndr	: Interceptors' airborne endurance.
airng	: Air intercept radar maximum range.
aienv	: Air intercept radar detection envelope.
aam	: Number of Air-to-Air missiles.
aamrng	: Air-to-Air missile range.
aamenv	: Air-to-Air missile firing envelope.
aampk	: Air-to-Air missile probability of kill.
aawndr	: Early warning aircraft airborne endurance.
aewrng	: Early warning radar maximum range.

Figure 4.3 Aircraft Record

2. Organization and Format

As mentioned earlier, Startem is the program in which the default database is presented, and Changem is the program in which the default database is used to form the game database. Figure (4.4) is a partial flow chart for Startem, where the entry flow path is determined by the selection from the first menu. The next selection menu provides a more detailed selection of how the game database is to be formed. This is done in the GAMECHOICES procedure. Figure (4.5) is a general flow chart for Changem, where the entry flow path is determined by the selection from the second menu. Notice it is not always necessary to pass through all the programs. From Intro it is possible to bypass most of Startem or to bypass Startem and Changem. Further, it is possible to bypass Changem from Startem. Actually, the only time a program is entirely skipped is when the operator chooses to play with a previously established database, thus skipping Startem and Changem.

3. Startem, Thesis1

The primary purpose of this program is to present the parameters of the default database; therefore, the largest portion of this program does just that. However, the entire program is executed, i.e., the default database is reviewed only if the first option, 'build a database with review', is selected from the menu in Intro. If the third option, 'use default database', is selected, this program simply transfers the default database files to the game database files. If the second option, 'build database, no review', is selected, this program presents the second selection menu, which provides a detailed selection of how the game database is to be built.

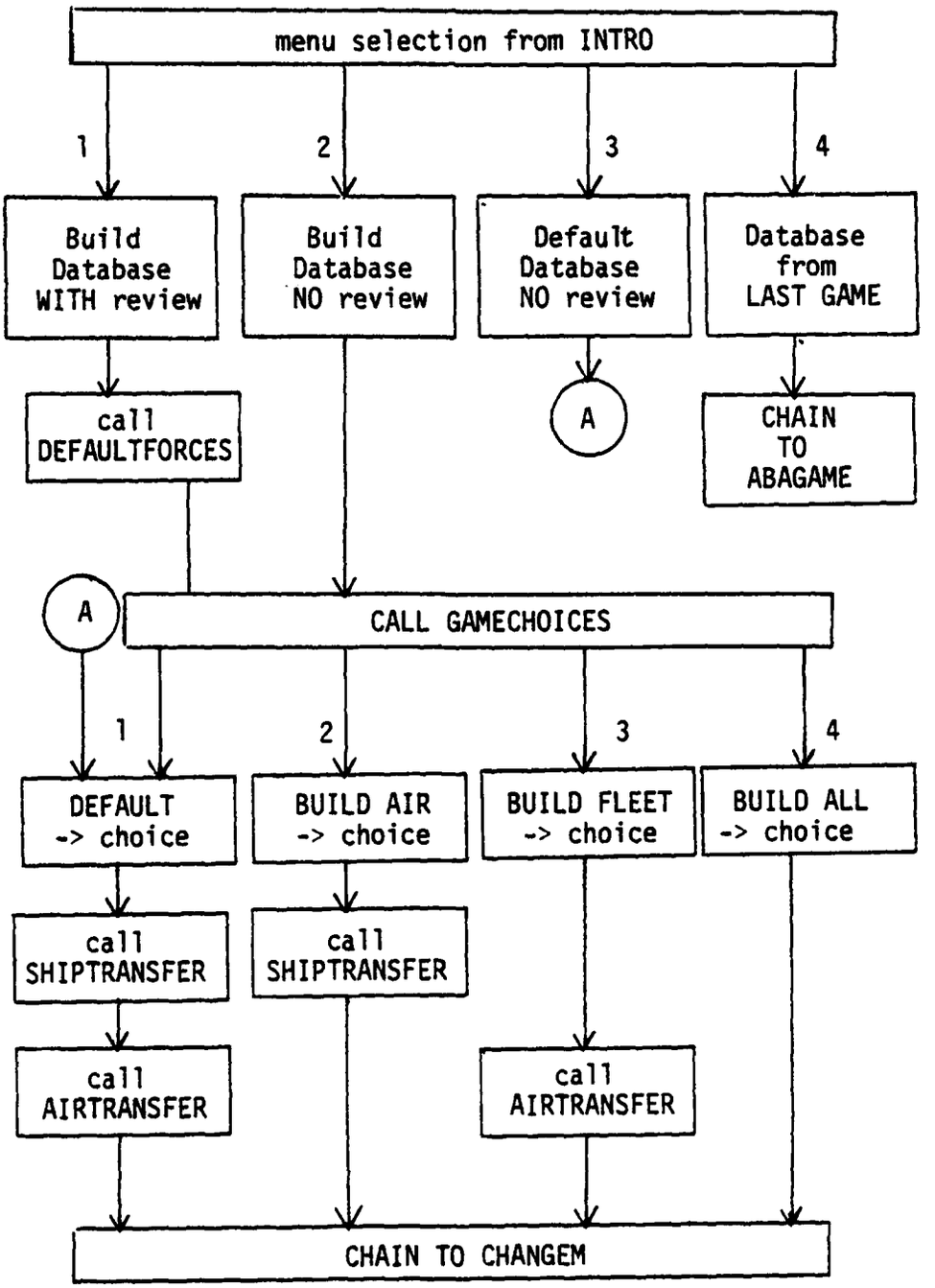


Figure 4.4 Startem Flowchart

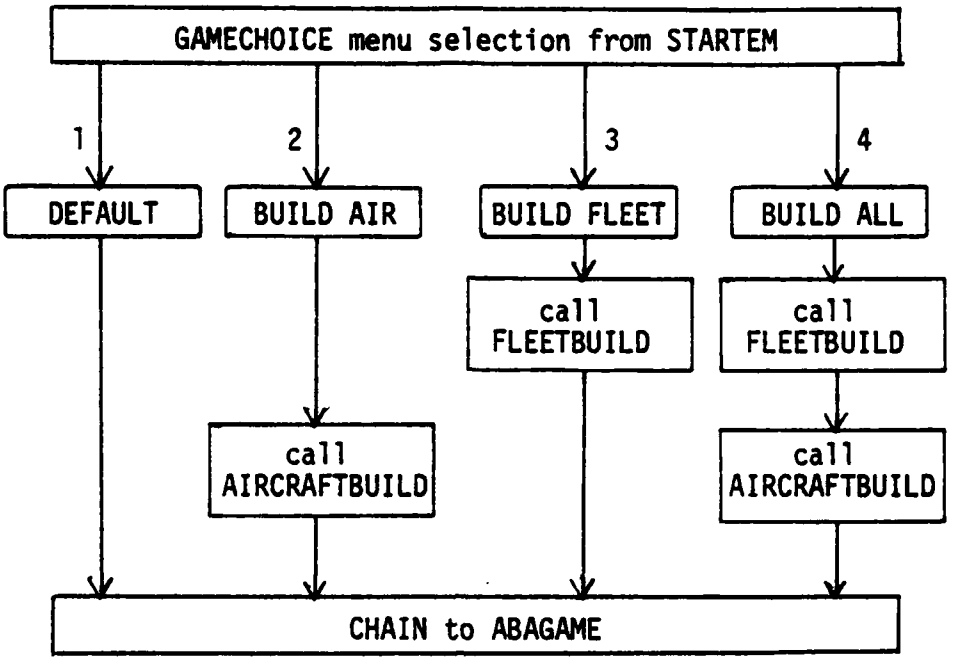


Figure 4.5 Changem Flowchart

The methodology of the review is similar to the chaining of the programs, except that this is from an individual program level. The main program calls the first procedure which starts a chain through four procedures. The order of presentation of the default database is;

i) A general overview of the ships, including aircraft on the carrier, type and number of missiles, and type of radars.

ii) A review of the Surface-to-Air missile parameters, and the ships' radar maximum ranges.

iii) A review of each ship's parameters, including position, course, and speed.

iv) A general overview of the aircraft involved and how they are deployed.

v) A review of individual aircraft parameters including position, altitude, course, groundspeed, armament, type radars, etc. Only airborne aircraft are listed in this review.

4. Changem, Thesis2

As indicated in Figure (4.5), Changem is where the game database is built. The presentation of the variables available for alteration is done in the same order as it was in Startem. If choice four, 'build fleet and aircraft', was selected from the second menu, then the ordering is virtually identical; otherwise, there is some distortion of the ordering.

> "Careful attention to the way the user sees a program - the so-called 'user interface' - ... makes the difference between programs that are friendly, forgiving, conversational, and humane and others that are hostile, rigid, obscure, and machine-like." [Ref. 4, p. 138]

There has evolved recently some general ideas as to the nature of interactive computer models, and how they should be designed with respect to the user. [Ref. 7] Several of these ideas, specifically feedback, consistency and simplicity, and how they have been incorporated in Changem are discussed below.

The user needs to be provided feedback for his actions. It is natural for the user to need to know that his actions have been understood and accepted. This feedback should be obvious and displayed where the user expects to see it. Changem echoes all user inputs on the screen directly left of where the cue to enter has occurred.

This presents another point, consistency. The user should not be required to guess where the cues will appear or where his feedback will occur. This consistency should be carried over from one aspect of

a program's operation to the next. This idea is evident throughout the program where a response to a program generated question is required.[4]

Simplicity. The simpler a program is to use the more it will be used. This also allows the more inexperienced user to use it correctly, competently and constructively.

As mentioned earlier, a primary concern of an interactive computer program is the data entry. Toward this end Changem and all other parts of the "whole" program use a consistent data entry process. In each instance the following sequence of steps occur:

i) prompt,

ii) provide feedback,

iii) perform error check,

iv) accept data entry.

A prompt is provided for each data entry. It is always as brief and as specific as possible. If the entry is to change a default value, this value is presented. If there is a length limit, the length of the entry is indicated by an underline of appropriate length. On entering the data, whether it is a Yes/No response or a numerical data entry, feedback is presented, immediately to the left of the prompt line.

An error check of the entry is performed. If the entry requires a single key stroke, as in answering a Yes/No question, then for an illegal response the entry is not accepted and the user is asked to reenter the response via an error message that again reflects what

---

[4]One aspect of program consistency was altered in the graphic displays. The method of continuing the program was purposely altered to a different key stroke than found elsewhere. An explanation follows in Chapter 5, Abagame.

is expected. If the entry is to be numerical, as in changing the X position of a unit, the program accepts only digits and characters acceptable in numbers; it will not accept anything else, i.e., it will accept a "+" or "-" as the first character and if the entry is a real number, a "." is accepted. If an integer value is expected, the "." will not be accepted anywhere in the entry.

V. ABAGAME, THESIS3

A. INTRODUCTION

This chapter explains the game portion of the program. It describes the main program and each of the major procedures.

B. MAIN PROGRAM

Figure (5.1) is a flow diagram for the main program of Abagame. When this program is called the game database has been built, either in a previous session or in the Changem program. Abagame first calls the procedures INITIALIZE and DOAIRLISTS which form the linked lists from the game database, and then SHOWFORMS is called which presents a display of the forms used in the graphic displays. The program then asks the user whether he wishes to have the computer step through the game at a fixed time step. If so, the program will not call the SHOWSTATUS, NEXTEVENTS or NEXTSTEP procedures while all other aspects of the program are identical.

Abagame then begins the loop that repeats each of the following procedures until the "game time" is greater than the default "endtime" of 160 minutes or the user tells the program to stop in NEXTSTEP. The first procedures called are AIRADARCNTC and SHIPRADARCNTC, which determine if any of the fleet units have any radar contacts. Then DISPLAGAME is called and presents the graphic displays of the fleet and the attack. SHOWSTATUS is called next and presents status information on the fleet units. The procedures NEXTEVENTS and SELECTOR, discussed

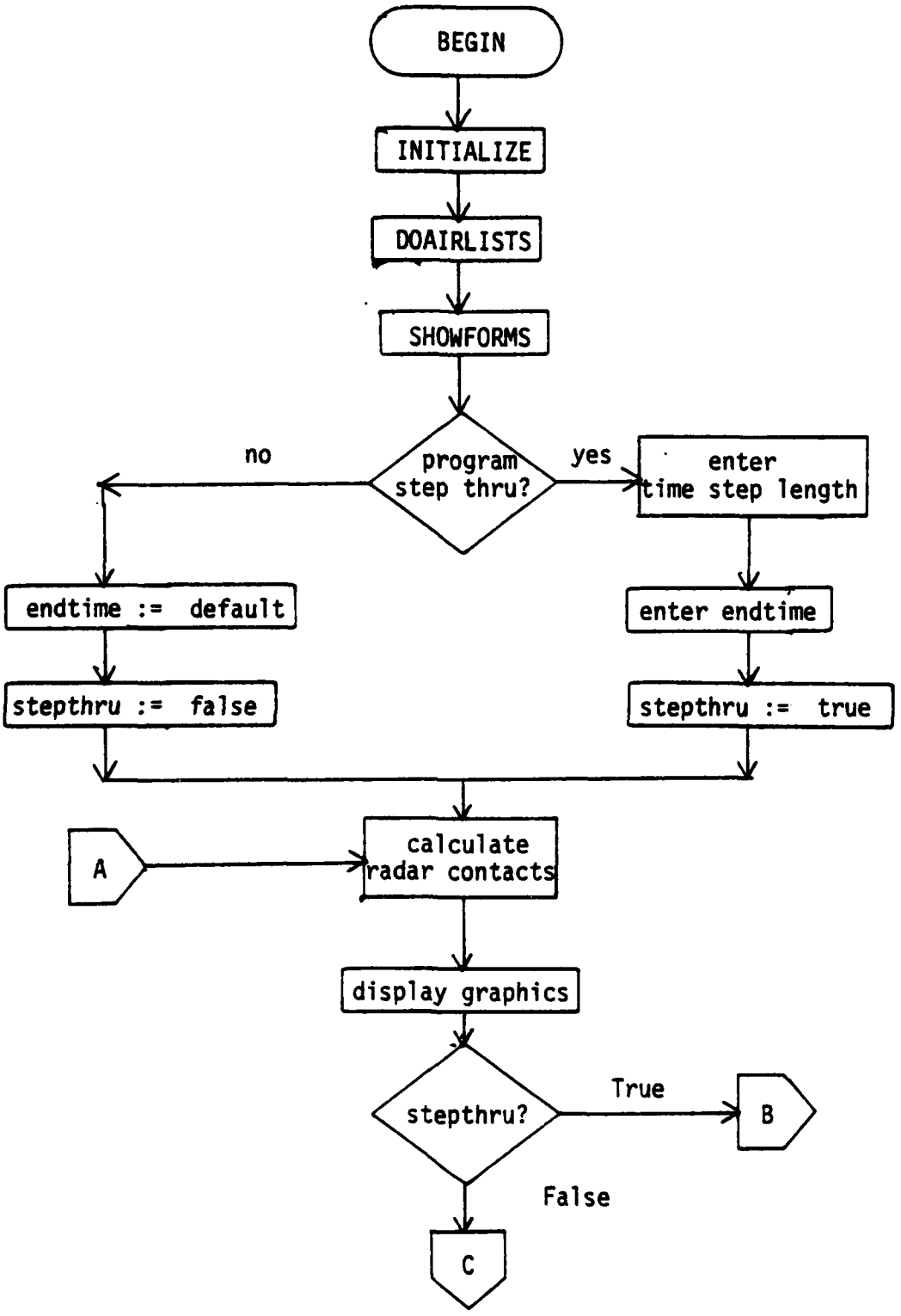


Figure 5.1 Abagame Main Program Flowchart

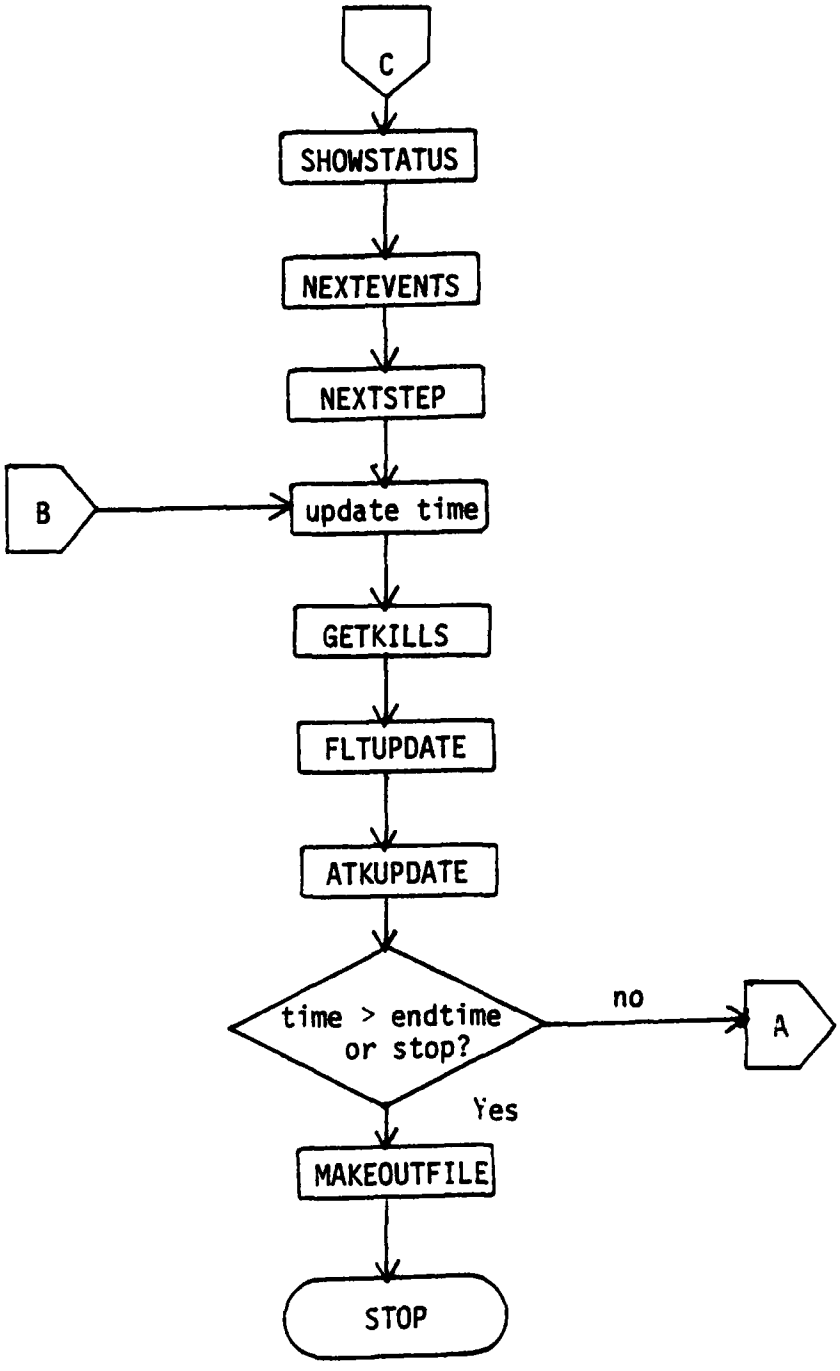


Figure 5.1 Continued

here as a single entity are called next. They allow the user to selectively alter the headings and speeds of friendly units. NEXTSTEP is called which allows the user to stop the program or continue with a time step of his choice. The program time is then updated according to the new time step. GETKILLS, the procedure which eliminates the "killed" enemy aircraft or "sunk" ships, is called next. The last procedures called in the loop are FLTUPDATE and ATKUPDATE. These two procedures update the fleet and attack positions according to the time step and the headings and speeds of the units. When the loop is exited, the program calls MAKEOUTFILE which creates "outfiles" of the surviving aircraft and ships.

C. INITIALIZE, DOAIRLISTS, SHOWFORMS

The procedures INITIALIZE and DOAIRLISTS are very similar. They form the linked lists that are used in the remainder of Abagame. Initialize is called first. It initializes the boolean variable "stop" to false, and the game time to zero. The SHIPDATA file is read from the game database and each record is assigned a sequential number and placed in the ship linked list. Also, the carrier's X and Y coordinates are noted. The carrier's position is referenced often throughout the program. DOAIRLISTS reads the AIRDATA file from the game database and according to whether the aircraft is an "enemy" or "frend" assigns it to the "atk" list or "air" list. The attack aircraft are assigned sequential numbers beginning at one. Since the displays are divided into attack and fleet, and the fleet consists of ships and friendly aircraft, the friendly aircraft need to be assigned non-conflicting numbers with the ships. Therefore, the friendly aircraft are assigned

sequential numbers beginning after the last number assigned a ship. This numbering is designed to facilitate the user's recognition of units between the graphic displays, the status displays, and the "event" displays.

SHOWFORMS is the procedure that shows the user the different forms that will be used in the graphic displays. This procedure first calls the procedures in the MAKEFORMS library unit that form the figures, and then displays them on a graphic display with appropriate explanation. The actual forms are made by activating or deactivating specific bits in two dimensional packed arrays of boolean variables.

D. GENERATING RADAR CONTACTS AND KILLS

The two procedures AIRADARCNTC and SHIPRADARCNTC along with MAKECNTC have several functions. They determine if an attack aircraft has been detected by one of the four radar types of the fleet or if it has been killed by a missile shot, either surface-to-air or air-to-air. Missile firings are done in an "uncoordinated" mode. If a contact is within the range and the firing envelope of a friendly unit that unit will fire. SHIPRADARCNTC also determines if one of the attack aircraft has successfully hit a ship with a missile or bomb and whether that ship has been sunk.

MAKECNTC forms a linked list of radar contacts, determining which attack aircraft are in radar contact and if more than one type radar is in contact which one will be displayed as having contact. MAKECNTC requires six parameters passed to it;

i) type of contacting radar,

ii) state of contact, either alive or killed,

iii) contact's X coordinate,

iv) contact's Y coordinate,

v) contact's number,

vi) number of unit holding contact.

AIRADARCNTC determines the interactions between friendly aircraft and attack aircraft. The overall process is to scan the friendly aircraft list as an outer loop and for each airborne friendly aircraft scan the attack aircraft list. For every combination of airborne friendly aircraft and each attack aircraft the ground distance, as opposed to slant distance, and the radar horizon are calculated. Radar horizon is calculated using;

rh := 1.25 * Sqrt(alt of a/c#1) + Sqrt(alt of a/c#2).

If the friendly aircraft is an early warning aircraft, then the distance between the units is checked. If this distance is less than the radar horizon and the maximum AEW radar range, then the procedure MAKECNTC is called, utilizing; 'air search radar', and 'contact is alive'. After return from MAKECNTC, the next attack aircraft on the list is checked through this entire process.

If the friendly aircraft is an interceptor, the distance between the units is compared to the air-to-air missile (AAM) maximum range, and the radar horizon. If the distance is less than these values, the target is within the firing envelope, and the interceptor is armed with missiles then a missile is fired. The interceptor's missile count is then decremented and a random number is generated in the RANDOM procedure of the APPLESTUFF unit of the system.library. RANDOM generates a psuedo-random number uniformly distributed between

zero and "maxint", the maximum integer represented in the APPLE III. If the random number is less than or equal to "maxint" multiplied by the probability of kill for the missile then the attack aircraft is declared killed and MAKECNTC is called with; 'air intercept radar', and 'contact is killed'. If the random number was greater than the "maxint" multiplied by the missile 'pk', then MAKECNTC is called with the same parameters except that 'contact is alive' is used. After return from MAKECNTC, the next attack aircraft on the list is checked through the process. However, if the last check (distance compared to missile maximum range, distance compared to radar horizon, target within firing envelope, and number of missiles greater than zero) was not true then the distance is compared to the air intercept radar maximum range, the radar horizon, and the air intercept radar detection envelope. If the distance is within both of these ranges and the target is within the detection envelope, then MAKECNTC is called with 'air intercept radar', and 'contact is alive'. If one of these comparisons is not true, then the next attack aircraft on the list is checked through the process. The procedure is exited when all aircraft have been paired for the comparisons.

In essence, this process is; check the shortest missile range; if within range, check for kill; if not within range, check for next longest radar range; if within range, generate contact; and if not within range, consider the next aircraft.

As indicated, SHIPRADARCNTC executes virtually the same process, the algorithm is identical, only the complexity is changed. Radar Horizon is calculated using:

rh := 1.25 * Sqrt(alt of a/c).

Ships may differ in their armament, but they are assumed by the program to have the same capabilities; short range surface-to-air missiles (SRSAM), long range surface-to-air missiles (LRSAM), fire control radar, and ship search radar. The SRSAM ranges (minimum and maximum) are compared to the distance between the ship and attack aircraft first, then LRSAM ranges, then fire control radar, then ship search radar. If the test that is true is for SRSAM, LRSAM, or the fire control radar then the radar parameter passed to MAKECNTC is 'fire control' otherwise it is 'ship search'. If the check was for a missile type then a random number is generated and a comparison similar to the one explained earlier is made. If,

random number <= maxint * missile pk,

is true then 'contact is killed' is passed to MAKECNTC, otherwise 'contact is alive' is passed.

At this point in SHIPRADARCNTC, it is determined if the attack aircraft has come within the range for a bomb drop or within range of an air-to-surface (ASM) missile. Range for a bomb drop is checked first, then the range for an ASM shot. A successful hit (determined by the same method as above) causes the bomb hit total or missile hit total to be incremented by one. The total hits on the ship are then compared to the hit tolerance for the ship and if the tolerance is exceeded, the ship is declared sunk. At this point, the next attack aircraft on the list is checked through this entire process. The procedure is exited when all ships have been compared with all attack aircraft.

MAKECNTC is the procedure that forms the radar contact linked list for each time step of the game. This procedure scans the contact list comparing each element's X position and Y position with the positions passed it by AIRADARCNTC or SHIPRADARCNTC to determine if this attack aircraft being passed is already on the list. If it is not on the list, then it is put there. If this aircraft is on the list and if the contact on the list is dead, then nothing occurs. If the incoming contact is indicated killed, then the incoming parameters replace those on the contact list. If the contact on the list is not dead already or indicated dead by the incoming parameters, the final contacting radar is determined according to the radar hierachy: fire control, air intercept, ship search and air search. As an example, when a fire control radar and an air intercept radar are in contact and the target is alive, then the fire control radar is declared the radar in contact, the displays will exhibit the fire control symbol and the status report shows fire control as the radar in contact. However, if the same radars hold contact and the interceptor had scored a kill, then the air intercept radar is used as the radar in contact. If the ship and interceptor fired a missile and both indicated a kill, then the one placed on the contact list first is used as the unit in contact. In this case, it would be the air intercept radar because AIRADARCNTC procedure is executed before SHIPRADARCNTC.

E. DISPLAGAME AND GRAPHICS DISPLAYS

Figure (5.2) is a general flow diagram for DISPLAGAME. DISPLAGAME is the procedure from which both the fleet and attack deployment displays are called. The first thing done is to call POSITRANSFER,

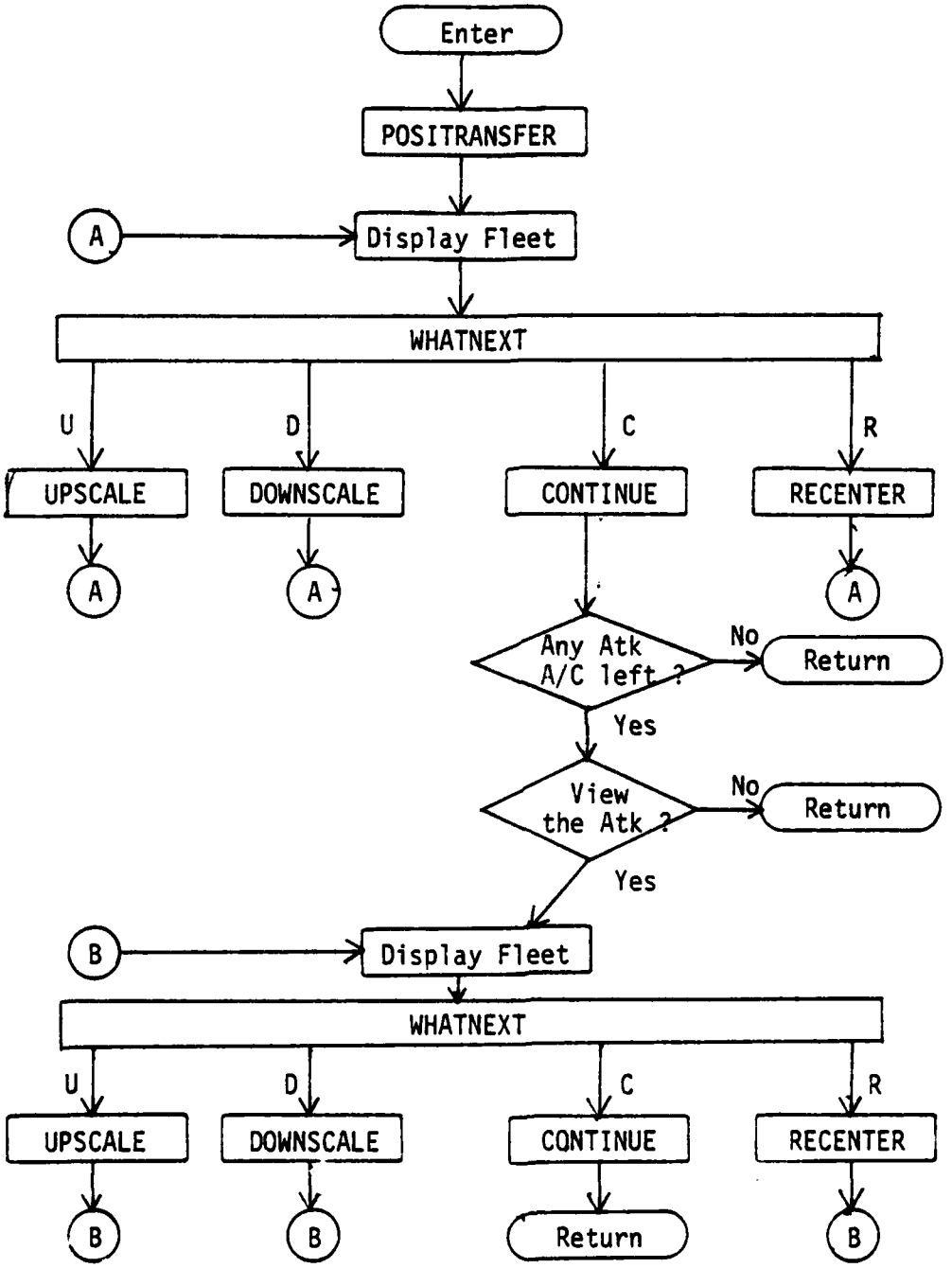


Figure 5.2 DISPLAGAME Flowchart

which is a procedure to build the two linked lists that the procedures SHOWFLEET and SHOWATTACK will use to make the displays. The first list is the "flt" list and the second is the "ene" list. Each is a linked list of records. Figure (5.3) illustrates these records, which are defined in the system.library unit GRAFSTUFF.

The "flt" list is formed by scanning the ship list and transferring the required information, then scanning the "air" list (friendly aircraft) and, provided the aircraft is airborne, transferring the required information. The "cnt" list (radar contact) is next scanned and the information on it is transferred. Likewise, the "ene" list is formed from the "atk" list (enemy aircraft). These separate lists are needed because the coordinate positions are actually changed when a screen is recentered, upscaled or downscaled.

```
flt = record
        link : fltpntr;
        what : fltype;
        num  : 0..255;
        xpos : real;
        ypos : real;
      end;

ene = record
        link : enepntr;
        what : enetype;
        num  : 0..255;
        xpos : real;
        ypos :  real;
      end;
```

Variable Meanings:

```
link : A pointer variable used in the linked lists.
what : A program type, used to determine what figure
       to use in the display.
num  : The unit number, assigned in Abagame.
xpos : The X coordinate position on the screen.
ypos : The Y coordinate position on the screen.
```

Figure 5.3 Display Records

The procedures SHOWFLEET and SHOWATTACK do just what their names imply. They scan the appropriate list ('flt' or 'ene') and draw the correct figure and unit number at the X-Y coordinate position. The procedure WHATNEXT is then called and presents a selection menu, consisting of four choices that are drawn on the graphic force display. The user can upscale or downscale the display or recenter on one of the figures on the display or he can continue the program.[5]

The same menu is provided on both the fleet and attack displays. The user is presented the fleet display on every time step through the game. After he sees this display, he is presented the option of viewing the attack display.

The three procedures SCALEUP, SCALEDOWN, and RECENTER are in the thesis3.-library unit GRAFSTUFF. Additionally, this unit defines the records that form the "flt" list and "ene" list. The scaling procedures are very similar in style and operation. The scale is either doubled or halved with each call to the respective procedure. The procedure RECENTER works in the following manner. The appropriate list is scanned to find the unit on which the display is to be recentered. Then adjustment figures for each coordinate plane are calculated that are the distance from the screen's center to the unit. Then the list is again scanned and each unit's X and Y positions are redefined according to the adjustment figures.

---

[5]This is the inconsistency mentioned earlier. This was done because the type ahead buffer of the APPLE III stores key strokes until the computer can act on them. During a recenter operation, a three digit number is allowed. Often though, the recenter is done on a single or double digit unit number thus a space or carriage return is required to enter the number. If this key is inadvertently held too long, it would cause the screen to recenter and then immediately type the space character. With the method used elsewhere, this would cause a page continuation. It was concluded this inconsistency was more desirable because it made the program a bit more "goof proof".

F. STATUS REPORTS, NEXTEVENTS, NEXTSTEP

The displays are followed by the menu for selecting the status reports of the fleet. These reports show the display number, type of unit, coordinate position, position relative to carrier (aircraft and radar contacts only), heading of unit, and speed of unit. These reports are amplifying information for the displays.

When the user is through reviewing the status reports, he is presented the menu for selecting the events he desires to take place in the next time step. When a selection is made from the event menu, the selection is followed by further questioning to determine what the user desires to accomplish. For example, if he chooses to move a fighter, he is presented with each airborne fighter and asked if this is the one he wishes to move.[6] If he answers no to each one, he is taken back to the menu and no changes are made. If he answers yes for one, he is asked to enter the new heading, velocity and altitude he desires for the unit. After this, he is presented with the event menu again. He can then make another selection or repeat a selection for another aircraft (or ship) or quit. The user is able to review the displays and/or the status reports from this menu. Reviewing the status reports allows the user to see the alterations he has made for the next time step.

When the user is satisfied with his actions and is ready to continue, he quits this menu and is presented the "next step" choices.

---

[6]This is actually a misnomer. The user alters the heading, speed and altitude of a unit and with the next update and display the units are "moved" accordingly.

He is told the current game time, in minutes, and is asked if he wants to stop the program. If he answers no, he is asked to enter the next time step increment. The last part of the program loop is the set of procedures that update the positions of the fleet and attack.

G. GETKILLS, FLTUPDATE, ATKUPDATE

The final three procedures in the program loop update the linked lists for the next loop. GETKILLS serves two functions, it deletes "sunk" ships from the ship list and deletes "killed" attack aircraft from the attack list. First, it scans the contact list checking for "killed" contacts. Then for each "killed" contact, it scans the "atk" list looking for the aircraft that has matching coordinate positions, and then deletes this aircraft from the "atk" list. When the scan of the contact list is completed and all "killed" attack aircraft are deleted, the ship list is then scanned. Each "sunk" ship is then deleted from the list. At the completion of GETKILLS, the lists contain only alive aircraft and floating ships.

FLTUPDATE is then called and updates the positions of the fleet units. The new ship list is scanned and the library procedure GETNEWXY is called for each ship. If the ship is the carrier, then the coordinate positions are noted for future game reference. The procedure then scans the "air" list and determines endurance for each airborne friendly aircraft. If the aircraft does not have enough endurance to last the time step, then it is deleted from the list. If the aircraft has enough endurance, then the endurance time is decremented and the aircraft's new coordinate positions are calculated and recorded by

GETNEWXY. If the aircraft is not airborne, then the aircraft's coordinate positions are set equivalent to the carrier's.

ATKUPDATE is the last procedure called in the loop of the program. ATKUPDATE scans the "atk" list and calculates and records the new coordinate positions for each attack aircraft. It next implements the attack profile. If the game time is greater than the attack retreat time, then the list is scanned again and the retreat heading, velocity and altitude are placed in the appropriate variables of each aircraft's record. If the game time is less than the retreat time, then game time is checked against a "look" time. This "look" time is considered to be an intelligence update time, i.e., after every multiple of this look time increment, the attack "gets an intelligence update", which consists of the new heading to the fleet. When the game time increases beyond these "look" intervals, the heading of each attack aircraft is updated. Also, when the first attack aircraft gets within 200 nautical miles of the carrier, each aircraft's altitude is changed to the "inbound" altitude. Most of these "profile variables" are written into the program as constants and therefore this "profile" can be changed by changing these constant values. Figure (5.4) is a list of the game constants and their meanings.

When this procedure is completed, the program checks for a user declared "stop" or a game time greater than the default "endtime". If either is true, the program exits the loop and calls the procedure that forms the outfiles of the program. These files are placed on the second disk and can be printed using the FILECHCKER program. The files consist of information concerning the forces still in "action" when the program was halted. If the program is not halted, the game is repeated beginning with the check for radar contacts.

```
retrtime =   120 : Game time of retreat.
retrthdg =   090 : Retreat heading.
retrtalt = 20000 : Retreat altitude
retrtvel =   400 : Retreat velocity
inbdalt  =   200 : Inbound attack altitude.
inbdist  =   200 : Distance from carrier for attack
                   altitude descent.
incr     =    20 : Intelligence update interval.
tol      =   0.5 : Miss tolerance for bomb drops.
hittol   =     7 : Number of hits a ship can endure.
recov    =     5 : Instant recovery radial distance.
tmdefault =  160 : Game end default time.
```

Figure 5.4 Abagame Program Constants

VI. FUTURE DEVELOPMENTS, APPLICATIONS

A. EXPANSION AND EXTENSIONS

There are a few areas of the program that could be enhanced to make the program more flexible, more realistic, or operate/use memory more efficiently.

Flexibility could be enhanced with the addition of a larger database or maybe several different default databases. More files could be created with several different types of scenarios, e.g., convoys, or offensive attacks against an air defense. The scenario of this program has much room for expansion. For example, the attack force could be altered for:

i) the azimuth of the attack,

ii) more or fewer aircraft,

iii) highly specialized aircraft,

iv) different formations,

v) time delays between attack aircraft, etc.

It would be a trivial matter to change the menus and chaining operations of Intro and Startem to allow more databases/scenarios. These changes could be instituted as possible classroom projects in a wargaming or simulation course. With the inclusion of these ideas into the program, the algorithms certainly would be scrutinized and assuredly enhanced and streamlined, thus enhancing efficiency. The program was developed and written with "brute force" algorithms, and therefore lacks algorithmic finesse. More time and a concentrated effort in this direction could lend more realism and/or increase the efficiency.

Several times throughout the development of this thesis, the problem of memory space limitations has occurred. Upgrading the computer to the 256K configuration would allow a more expanded and realistic solution to the presentations. This expansion would allow more drivers in the system definition and more memory for the program. The addition of a "printer" driver would enable printed output, from tables and lists to a complete listing of the default or game database. The program does create "outfiles" that retain information on and the status of the surviving ships and aircraft. The textfile/program FILECHCKER will print these files to a printer if the system is coupled to an rs232 port.[7] Filechcker requires compilation and a system configured for a "printer" driver before it can be run.

Another avenue of expansion that could increase the program flexibility would be the addition of several attack profile (ATKUPDATE) procedures. If this were done, the user could be presented with a menu for selecting the profile he desires to implement at the start of the Abagame program, then based on his selection the appropriate procedure would be called when the attacking force's positions are to be updated.

An enhancement that would require extensive changes to the basic structure, but could add a higher level of realism would be the incorporation of an event type structure between the time steps. With this game structure, a more realistic approach to radar detections and missile

---

[7]This can be changed simply. Enter the text version of FILECHCKER and change the string variable in the rewrite statement from .rs232 to .silentype or .printer or the appropriate name of the "printer" device driver.

firing envelopes and of missile interceptions could be calculated and implemented. This would require more elaborate mathematical models. Specifically involving missile and aircraft interaction geometries.

Alternatively, all interactions could be calculated on a very short time step structure between the game "review" times of the user supplied time step. This would require less extensive program changes than the next event structure and it would help eliminate the occurrence of missed interactions because of long time steps.

B. APPLICATIONS

The air battle computer program is not designed to test one's knowledge of weapon systems or characteristics; it is helpful in ascertaining the effects of tactical employment of various types of weapon systems. It illustrates the consequences of decisions and of different courses of actions on the many possible interactions between adversary units and provides an easily understood display of movements and operations of the ships and aircraft. The air battle program is most helpful with the timing of the tactical decisions required in an air battle. The ephemeral nature of decisions in a tactical situation belies the importance these decisions hold on the outcome of a battle. The length of time one decision affects the battle is often very short; however, the entire outcome of the battle may rest entirely on one of these tactical decisions.

This program cannot be considered a cornucopia of solutions to the tactical decision problem areas it presents. It lacks realism in very important aspects of unit characteristics and the solution of interaction geometries. It provides, however, a learning experience in a recreational

environment and an interesting and economical method for exploring tactical environment decisions. Repeated use of the program is required to appreciate the subtle differences between one manner of tactical decision implementation and another.

PROGRAM LISTING

```
UNIT THESISTUFF;
INTRINSIC code 24 data 25;

                              INTERFACE

  TYPE
    shipptr    = ^ship;
    airptr     = ^aircraft;
    cntptr     = ^contact;
    shiptype   = (cv,dest,crsr);
    frndtype   = (intert,aew);
    ifftype    = (frend,enemy);
    radartype  = (asrch,ssrch,air,fcon);
    cntrl      = (clr,off,on,bel,bs,ht,lf,clrln,clrps);
    contact    = record
                   link    : cntptr;
                   who     : radartype;
                   num     : 0..255;
                   rdnm    : 0..255;
                   xpos    : real  ;
                   ypos    : real ;
                   dead    : boolean;
                 end;
    ship       = record
                   link    : shipptr;
                   class   : shiptype;
                   num     : 0..255;
                   xpos    : real  ;          { X coordinate position   }
                   ypos    : real  ;          { Y coordinate position   }
                   rim     : 0..359;          { 360 = 000 = North       }
                   spd     : 0..30;           { Speed of advance        }
                   fcrng   : 0..255;          { Range of fire control   }
                   ssrng   : 0..255;          { Range of search radar   }
                   lrsam   : 0..255;          { # of long range SAM     }
                   srsam   : 0..255;          { # of short range SAM    }
                   lrmpk   : real   ;         { LR SAM prob. of kill    }
                   srmpk   : real   ;         { SR SAM prob. of kill    }
                   srmin   : 0..255;          { SR minimum target dist  }
                   srmax   : 0..255;          { SR maximum target dist  }
                   lrmin   : 0..255;          { LR minimum target dist  }
                   lrmax   : 0..255;          { LR maximum target dist  }
                   mhits   : 0..255;          { Nuber of missile hits   }
                   bhits   : 0..255;          { Number of bomb hits     }
                   sunk    : boolean;         { Indication of total hits}
                 end;
     aircraft  = record
                   link    : airptr;
                   num     : 0..255;
                   xpos    : real  ;
                   ypos    : real  ;
                   alt     : 0..30000;
                   azmth   : 0..359;
                   velcty  : 0..2000;
                   iff     : ifftype;
```

```
                case iffltype of
                  enemy :
                    (       asm     : 0..31;
                            asmrng  : 0..255;
                            asmenv  : 0..359;
                            asmpk   : real    ;
                            bomb    : 0..31;
                            bombpk  : real              );
                  friend :
                    (       acfrnd  : frndtype;
                      case frndtype of
                            intcpt :
                              (       intndr : 0..255;
                                      airng  : 0..255;
                                      aienv  : 0..359;
                                      aam    : 0..31;
                                      aamrng : 0..255;
                                      aamenv : 0..359;
                                      aampk  : real     );
                          aew    :
                              (       aewndr : 0..255 ;
                                      aewrng : 0..255   ));
              end;                  {   record          }

VAR
  selection : char;
  controlnum : 0..31;

PROCEDURE SCRNCNTRO ( dowhat : cntro ) ;
PROCEDURE CONTINUE;
PROCEDURE YESNOSEL;
FUNCTION  CHANGEIT : boolean;
FUNCTION  READINT  ( lnth : integer ) : integer;
FUNCTION  READREAL ( lnth : integer ) : real;
                        IMPLEMENTATION

PROCEDURE SCRNCNTRO;

  begin
    case dowhat of
      bel   :   controlnum := 07;       {   Sound Bell                  }
      bs    :   controlnum := 08;       {   Backspace Cursor            }
      ht    :   controlnum := 09;       {   Horizontal Tabulation       }
      lf    :   controlnum := 10;       {   Linefeed                    }
      off   :   controlnum := 14;       {   Screen Off                  }
      on    :   controlnum := 15;       {   Screen On                   }
      clr   :   controlnum := 28;       {   Clear Screen                }
      clrln :   controlnum := 29;       {   Clear Line From Cursor      }
      clrss :   controlnum := 31;       {   Clear Screen From Cursor    }
    end;        {   case    }
    unitwrite (1,controlnum,2,,12);
  end;                          {     SCRNCNTRO     }
```

```
PROCEDURE CONTINUE;

  begin
    write  ('         Press the SPACE BAR to continue.    ');
    read (selection);      screntro (on);
    while NOT (selection = ' ') do
      begin
        screntro(bel);      writeln;
        write  ('     You must hit the SPACE BAR !!  -- Please try again: ');
        read (selection);
      end;
  end;                                {      CONTINUE     }

PROCEDURE YESNOSEL;

  begin
    write ('      Select                 or   N(o      ');
    read (selection);       screntro (on);
    while NOT (selection in ['Y','y','N','n']) do
      begin
        screntro(bel);      writeln;
        writeln('     You must enter a (Y or y)  or  (N or n) !!      ');
        write  ('                                        Please try again :  ');
        read (selection);
      end;
  end;                                {      YESNOSEL     }

FUNCTION CHANGEIT;

  begin
    writeln;
    writeln('      Do you wish to change this value?    ');
    writeln;    yesnosel;     writeln;     writeln;
    case selection of
      'Y','y' : begin
                  changeit :- true;
                  write('         Enter the new value :  ');
                end;
      'N','n' : begin
                  changeit := false;
                  write('         Old value retained.... ');
                end;
    end;       {     case     }
  end;                                {      CHANGEIT    }

FUNCTION READINT;

  CONST
    bks =  8;
    cr  = 13;
    sp  = 32;

  VAR
    charray : array [1..10] of char;
```

```
    num : integer;
    posit : 1..10;
    neg : boolean;
    digits : set of char;

  begin                                 {     READINT   }
    digits := ['0'..'9'];
    for posit := 1 to lnth do  write ('_');
    for posit := 1 to lnth do  scruentro(bs);
    posit := 1;
    while posit = 1 do
      begin
        read (keyboard, charray [posit]);
        if (charray [posit] in digits + ['+','-'])
          then  begin
                  write (charray [posit]);
                  posit := posit + 1;
                end;
      end;          {     while     }
    while posit <= lnth  do
      begin
        read (keyboard, charray [posit]);
        if (charray [posit] in digits )
          then  begin
                  write (charray [posit]);
                  posit := posit + 1;
                end
          else  begin
                  if charray [posit] = chr(bks)
                    then  begin  scruentro(bs);  posit := posit - 1;  end;
                  if (charray [posit] in [chr(sp),chr(cr)])
                    then lnth := posit - 1;
                end;
      end;            {      while    }
    num := 0;
    if charray [1] = '-'  then  neg := true   else  neg := false;
    for posit := 1 to lnth do
      if (charray [posit] in digits)
        then num := 10 * num + ord (charray [posit]) - ord ('0');
    if neg
      then   readint := - num
      else   readint :=   num;
  end;                                  {      READINT    }

FUNCTION READREAL;

  CONST
    bks =  8;
    cr  = 13;
    sp  = 32;

  VAR
    charray : array [1..10] of char;
    num, divisor : integer;
```

```
    renum : real;
    posit, decimal, i : 0..10;
    neg : boolean;
    digits : set of char;

  begin                                 {     READREAL   }
    digits := ['0'..'9'];
    for posit := 1 to lnth do  write ('_');
    for posit := 1 to lnth do  screntro(bs);
    posit := 1;    decimal := 0;      divisor := 1;
    while posit = 1 do
      begin
        read (keyboard, charray [posit]);
        if (charray [posit] in digits + ['+','-','.'])
          then  begin
                  write (charray [posit]);
                  posit := posit + 1;
                end;
        if charray [posit] = '.'   then  decimal := posit - 1;
      end;          {     while     }
    while posit <= lnth  do
      begin
        read (keyboard, charray [posit]);
        if (charray [posit] in digits + ['.'])
          then  begin
                  write (charray [posit]);
                  posit := posit + 1;
                  if charray [posit] = '.'   then  decimal := posit - 1;
                end
          else  begin
                  if charray [posit] = chr(bks)
                    then  begin  screntro(bs);  posit := posit - 1;  end;
                  if (charray [posit] in [chr(sp),chr(cr)])
                    then lnth := posit - 1;
                end;
      end;            {     while    }
    num := 0;
    if charray [1] = '-'  then  neg := true   else  neg := false;
    for posit := 1 to lnth do
      if (charray [posit] in digits)
        then num := 10 * num + ord (charray [posit]) - ord ('0');
    if (decimal <> 0)  and  (lnth <> decimal)
      then for i := 1 to (lnth - decimal) do
             divisor := divisor * 10;
    renum := num / divisor;
    if neg
      then   readreal := -renum
      else   readreal :=  renum;
  end;                              {     READREAL     }

end.                                {     THESISTUFF     }
```

```
UNIT BEARINGS;
intrinsic code 34;

                              INTERFACE

  USES
    realmodes,transcend;

  CONST
    pi = 3.14159271;

  PROCEDURE GETNEWXY ( TM,HDG,VEL : INTEGER; VAR NX,NY : REAL);
  FUNCTION DEGREES   ( XO,YO,XR,YR : REAL) : INTEGER;
  FUNCTION DISTANCE  ( XO,YO,XR,YR : REAL) : INTEGER;

                            IMPLEMENTATION

  PROCEDURE GETNEWXY;
    VAR
      dist, rads : real;

    begin
      dist := vel * tm / 600;
      rads := hdg * pi / 180;
      nx   := nx + dist * sin (rads);
      ny   := ny + dist * cos (rads);
    end;

  FUNCTION DEGREES;

    VAR
      xdif, ydif : real;
      value      : integer;

    begin
      xdif := xo - xr;
      ydif := yo - yr;
      if xdif = 0
        then if ydif = 0
               then  degrees := 0
               else if ydif > 0
                      then degrees := 180
                      else degrees := 000
        else if ydif = 0
               then if xdif > 0
                      then degrees := 270
                      else degrees := 090
               else begin
                      value := round (180 / pi * atan (xdif / ydif));
                      if abs (value) <> value
                        then if xdif < 0
                               then degrees := 180 + value
```

```
                                    else degrees := 360 + value
                                else if xdif > 0
                                    then degrees := 180 + value
                                    else degrees := value
                            end;
    end;

  FUNCTION DISTANCE;

    VAR
      xdif, ydif : real;

    begin
      xdif := xu - xr;
      ydif := yu - yr;
      distance := round (sqrt ( sqr (xdif) + sqr (ydif)) );
    end;

  end.



UNIT MAKEFORMS;
intrinsic code 26 data 27;

                              INTERFACE

VAR
  shipform      : packed array [0..5,0..7] of boolean;
  sewform       : packed array [0..5,0..9] of boolean;
  infform       : packed array [0..5,0..9] of boolean;
  sewndr        : packed array [0..4,0..4] of boolean;
  fendr         : packed array [0..4,0..4] of boolean;
  dindr         : packed array [0..4,0..4] of boolean;
  ssndr         : packed array [0..4,0..4] of boolean;


PROCEDURE MISFORMS;
PROCEDURE AIRFORMS;

                              IMPLEMENTATION

PROCEDURE MISFORMS;

  VAR
    i,j : 0..20;

  begin
    for i := 0 to 5 do                          { SHIP FORM                 }
      for j := 0 to 7 do
        begin
          shipform [i,j] := true;
```

```
          if ((i = 0)  and  ((j < 3)  or (j > 4)))
            then   shipform [i,j] := false;
          if (((i = 1)  or  (i = 4))  and  ((j < 2)  or (j > 5)))
            then   shipform [i,j] := false;
        end;
      shipform [3,0] := false;
      shipform [3,7] := false;
      for i := 0 to 4 do                              { AIR SEARCH RADAR        }
        for j := 0 to 4 do
          aswrdr [i,j] := true;
      for i := 1 to 3 do
        for j := 1 to 3 do
          aswrdr [i,j] := false;
      for i := 0 to 4 do                              { INTERCEPTOR RADAR       }
        for j := 0 to 4 do
          if ((i = 2)  or (j = 2))
            then   airdr  [i,j] := true
            else   airdr  [i,j] := false;
      for i := 0 to 4 do                              { FIRE CONTROL RADAR      }
        for j := 0 to 4 do
          if ((i = j)  or (i + j = 4))
            then   fcrdr  [i,j] := true
            else   fcrdr  [i,j] := false;
      for i := 0 to 4 do                              { SHIP SEARCH RADAR       }
        for j := 0 to 4 do
          begin
            if odd(i + j)
              then   ssrdr  [i,j] := false
              else   ssrdr  [i,j] := true ;
            ssrdr  [0,0] := false;
            ssrdr  [0,4] := false;
            ssrdr  [4,0] := false;
            ssrdr  [4,4] := false;
          end;
    end;

PROCEDURE AIRFORMS;

  VAR
    i,j : 0..20;

  begin
    for i := 0 to 7 do                                { INTERCEPTOR/ATTACK FORM }
      for j := 0 to 9 do
        if i > 3 then  intform [i,j] := true   else  intform [i,j] := false;
    intform [0,6] := true;
    intform [0,7] := true;
    intform [1,3] := true;
    intform [1,6] := true;
    intform [2,4] := true;
    intform [2,3] := true;
    intform [3,3] := true;
    intform [3,4] := true;
    intform [2,9] := true;
```

```
    initform [3,8] := true;
    initform [3,9] := true;
    for i := 0 to 5 do                          { EARLY WARNING A/C FORM }
      for j := 0 to 9 do
        if ((i = 2)  or (i = 3))
          then aewform [i,j] := true
          else if ((j = 3)  or  (j = 4))
                 then aewform [i,j] := true
                 else aewform [i,j] := false;
    aewform [1,8] := true;
    aewform [1,9] := true;
    aewform [4,8] := true;
    aewform [4,9] := true;
  end;

end.



UNIT GRAFSTUFF;
intrinsic code 32 data 33;

                              INTERFACE

CONST
  xcen = 139;
  ycen = 95;

TYPE
  fltptr = ^flt;
  enemptr = ^ene;
  fltype  = (boat,early,fight,aewcnt,aicnt,fcont,scnt);
  enetype = (plane,cntc);
  flt     = record
              link : fltptr;
              what : fltype;
              num  : 0..255;
              xpos : real;
              ypos : real;
            end;
  ene     = record
              link : enemptr;
              what : enetype;
              num  : 0..255;
              xpos : real;
              ypos : real;
            end;

VAR
  fltbase, fltnext : fltptr;
  enebase, enenext : enemptr;
  stkref           : boolean;
  nocent           : integer;
```

```
PROCEDURE RECENTER;
PROCEDURE SCALEUP;
PROCEDURE SCALEDOWN;

                                IMPLEMENTATION

PROCEDURE RECENTER;

  VAR
      xadj,yadj : real;


    if atkstaf
      then begin
             enenext := enebase;
             while ((enenext^.num <> nucent)   and   (enenext <> nil)) do
                      enenext := enenext^.link;
             xadj := xcen - enenext^.xpos;
             yadj := ycen - enenext^.ypos;
             enenext := enebase;
             while enenext <> nil do
               begin
                 enenext^.xpos := enenext^.xpos + xadj;
                 enenext^.ypos := enenext^.ypos + yadj;
                 enenext := enenext^.link;
               end;
           end
      else begin
             fltnext := fltbase;
             while ((fltnext^.num <> nucent)   and   (fltnext <> nil)) do
                      fltnext := fltnext^.link;
             xadj := xcen - fltnext^.xpos;
             yadj := ycen - fltnext^.ypos;
             fltnext := fltbase;
             while fltnext <> nil do
               begin
                 fltnext^.xpos := fltnext^.xpos + xadj;
                 fltnext^.ypos := fltnext^.ypos + yadj;
                 fltnext := fltnext^.link;
               end;
           end;            {    if    }
  end;

PROCEDURE SCALEUP;

  VAR
    xadj,yadj : real;

  begin
    if atkstaf
      then begin
             enenext := enebase;
             while enenext <> nil do
```

```
                begin
                  xadj := xcen - enenext^.xpos;
                  yadj := ycen - enenext^.ypos;
                  enenext^.xpos := xcen - xadj / 2;
                  enenext^.ypos := ycen - yadj / 2;
                  enenext := enenext^.link;
                end;
            end
        else begin
               fltnext := fltbase;
               while fltnext <> nil do
                 begin
                   xadj := xcen - fltnext^.xpos;
                   yadj := ycen - fltnext^.ypos;
                   fltnext^.xpos := xcen - xadj / 2;
                   fltnext^.ypos := ycen - yadj / 2;
                   fltnext := fltnext^.link;
                 end;
             end;       <   if   >
  end;

PROCEDURE SCALEDOWN;

  VAR
    xadj,yadj : real;

  begin
    if atkdraf
      then begin
             enenext := enebase;
             while enenext <> nil do
               begin
                 xadj := xcen - enenext^.xpos;
                 yadj := ycen - enenext^.ypos;
                 enenext^.xpos := xcen - xadj * 2;
                 enenext^.ypos := ycen - yadj * 2;
                 enenext := enenext^.link;
               end;
           end
      else begin
             fltnext := fltbase;
             while fltnext <> nil do
               begin
                 xadj := xcen - fltnext^.xpos;
                 yadj := ycen - fltnext^.ypos;
                 fltnext^.xpos := xcen - xadj * 2;
                 fltnext^.ypos := ycen - yadj * 2;
                 fltnext := fltnext^.link;
               end;
           end;       <   if   >
  end;

END.
```

```
PROGRAM FILEMAKER;

  USES thesistuff;

  VAR
    shipinfile : file of ship;
    airinfile  : file of aircraft;
    i,j        : 0..200;

  PROCEDURE MAKESHIPFILE;

    begin
      with shipinfile^ do
        begin
          link     := nil;
          class    := cv;
          num      := 0;
          xpos     :=  5;
          ypos     := 10;
          pim      := 025;
          soa      := 20;
          fcrng    := 40;
          ssrng    :=  80;
          lrsam    :=  0;
          srsam    := 40;
          lrmpk    := 0.4 ;
          srmpk    := 0.7 ;
          srmin    := 0;
          srmax    := 20;
          lrmin    := 10;
          lrmax    := 40;
          mirls    := 0;
          birls    := 0;
          sunk     := false;
        end;          {       with      }
      put (shipinfile);
    for i := 1 to 4 do
      begin
        with shipinfile^ do
          begin
            link     := nil;
            class    := dest;
            num      := 0;
            pim      := 025;
            soa      := 20;
            fcrng    := 40;
            ssrng    :=  80;
            lrsam    :=  0;
            srsam    := 20;
            lrmpk    := 0.4 ;
            srmpk    := 0.7 ;
```

```
            srmin    := 0;
            srmax    := 20;
            lrmin    := 10;
            lrmax    := 40;
            mhits    := 0;
            bhits    := 0;
            sunk     := false;
            case i of
              1 : begin   xpos := 3;  ypos := 11;   end;
              2 : begin   xpos := 6;  ypos := 12;   end;
              3 : begin   xpos := 7;  ypos :=  9;   end;
              4 : begin   xpos := 4;  ypos :=  8;   end;
            end;                 { case }
          end;                   { with }
        put (shipinfile)
      end;                       { for }
    for i := 1 to 4 do
      begin
        with shipinfile^ do
          begin
            link     := nil;
            class    := crsr;
            num      := 0;
            pim      := 025;
            soa      := 20;
            fcrns    := 40;
            ssrns    :=  80;
            lrsam    := 20;
            srsam    := 20;
            lrmpk    := 0.4 ;
            srmpk    := 0.7 ;
            srmin    := 0;
            srmax    := 20;
            lrmin    := 10;
            lrmax    := 40;
            mhits    := 0;
            bhits    := 0;
            sunk     := false;
            case i of
              1 : begin   xpos := 2;  ypos := 13;   end;
              2 : begin   xpos := 5;  ypos := 14;   end;
              3 : begin   xpos := 8;  ypos := 13;   end;
              4 : begin   xpos := 9;  ypos := 10;   end;
            end;                 { case }
          end;                   { with }
        put (shipinfile)
      end;                       { for }
end;                             { MAKESHIPFILE }
```

```
PROCEDURE MAKEFRENDLYAIR;

  begin
    for i := 1 to 24 do
      begin
        with airinfile^ do
         begin
           link   := nil;
           xpos := 3;
           ypos := 10;
           alt  := 0;
           azmth  :=    0;
           velcty :=    0;
           iff    := frend;
           acfrnd := intcpt;
           intndr := 120;       { in minutes      }
           airng  := 40;
           aieny  := 120;
           aam    := 6;
           aamrk  := 0.33;
           aamrng := 20;
           aamenv := 60;
           case i of
             1 : begin
                   xpos   := 5;
                   ypos   := 10.3;
                   alt    := 20000;
                   azmth  := 025;
                   velcty := 25;
                 end;
             2 : begin
                   xpos   := 3.3;
                   ypos   := 10.3;
                   alt    := 20000;
                   azmth  := 025;
                   velcty := 25;
                 end;
           end;        {   case   }
         end;          {    with   }
       put (airinfile);
      end;             {    for    }
    for i := 1 to 6 do
      begin
        with airinfile^ do
          begin
            link   := nil;
            xpos   := 3;
            ypos   := 10;
            alt    := 0;
            azmth  :=    0;
            velcty :=    0;
            iff    := frend;
            acfrnd := aew;
            aewndr := 240;
```

```
            aewrng := 200;
            case i of
              1 : begin
                    xpos   :=  0;
                    ypos   := 23;
                    alt    := 15000;
                    azmth  := 025;
                    velcty := 25;
                  end;
              2 : begin
                    xpos   := 12;
                    ypos   := 26;
                    alt    := 15000;
                    azmth  := 025;
                    velcty := 25;
                  end;
              3 : begin
                    xpos   := 19;
                    ypos   := 14;
                    alt    := 15000;
                    azmth  := 025;
                    velcty := 25;
                  end;
            end;      { case }
          end;        { with }
        put (airinfile);
      end;            { for }
end;                          { MAKEFRENDLYAIR }

PROCEDURE MAKEATTACKAIR;

  begin
    for i := 1 to 6 do
    for j := 1 to 3 do
      begin
        with airinfile^ do
          begin
            link   := nil;
            azmth  :=  270;
            iff    := enemy;
            asmrk  := 0.60;
            asmenv := 180;
            aswrng := 30;
            bumbrk := 0.55;
            case i of
              1 : begin
                    if j = 1
                      then  begin    xpos := 50.0;   ypos := 30.0; end
                      else if j = 2
                             then begin xpos := 50.5; ypos := 30.5; end
                             else begin xpos := 50.5; ypos := 29.5; end;
                    alt    := 20000;
                    velcty :=  350;
                    asm    := 6;
```

```
      bomb   := 2;
    end;       {   case 1   }
2 : begin
      if j = 1
        then   begin    xpos := 60.0;   ypos := 40.0; end
        else if j = 2
               then begin xpos := 60.5; ypos := 40.5; end
               else begin xpos := 60.5; ypos := 39.5; end;
      alt    := 10000;
      velcty :=  400;
      asm    := 6;
      bomb   := 0;
    end;       {   case 2    }
3 : begin
      if j = 1
        then   begin    xpos := 60.0;   ypos := 30.0; end
        else if j = 2
               then begin xpos := 60.5; ypos := 30.5; end
               else begin xpos := 60.5; ypos := 29.5; end;
      alt    := 10000;
      velcty :=  400;
      asm    := 6;
      bomb   := 0;
    end;       {   case 3    }
4 : begin
      if j = 1
        then   begin    xpos := 60.0;   ypos := 20.0; end
        else if j = 2
               then begin xpos := 60.5; ypos := 20.5; end
               else begin xpos := 60.5; ypos := 19.5; end;
      alt    := 10000;
      velcty :=  400;
      asm    := 6;
      bomb   := 0;
    end;       {   case 4    }
5 : begin
      if j = 1
        then   begin    xpos := 60.0;   ypos := 35.0; end
        else if j = 2
               then begin xpos := 60.5; ypos := 35.5; end
               else begin xpos := 60.5; ypos := 34.5; end;
      alt    := 10000;
      velcty :=  350;
      asm    := 4;
      bomb   := 2;
    end;       {   case 5    }
6 : begin
      if j = 1
        then   begin    xpos := 60.0;   ypos := 25.0; end
        else if j = 2
               then begin xpos := 60.5; ypos := 25.5; end
               else begin xpos := 60.5; ypos := 24.5; end;
      alt    := 10000;
      velcty :=  350;
```

```
                    asm    := 4;
                    bomb   := 2;
                  end;          { case 6    }
              end;              { case      }
            end;                { with      }
          put (airinfile)
        end;                    { for/for   }
    end;                          { MAKEATTACKAIR    }

  begin                           { MAKEFILES      }
    rewrite (shipinfile,'np2:shipinfile.data');
    rewrite (airinfile ,'np2:airinfile.data');
    makeshipfile;
    makefriendlyair;
    makeattackair;
    close (shipinfile,lock);
    close (airinfile,lock);
  end.                  { MAKEFILES      }



PROGRAM FILECHCKER;

  USES  thesistuff;

  VAR
    shipoutfile  : file of ship;
    airoutfile   : file of aircraft;
    i,j      : 0..200;
    kind     : string;
    outfile  : text;

  PROCEDURE SHIPCHECK;

    PROCEDURE SHIPWRITE;

      begin
       with shipoutfile^ do
        begin
         writeln(outfile);    writeln(outfile);    writeln(outfile);
         writeln(outfile,'     ',kind,' number ',num:2,'.');
         writeln(outfile);
         write  (outfile,'X-Y coordinate position         : (');
         writeln(outfile,xpos:4:1,',',ypos:4:1,').');
         writeln(outfile,'Current direction of movement   : ',dim:3,'.');
         writeln(outfile,'Current speed of advance        : ',soa:3,'.');
         writeln(outfile,'Number of long range SAM''s      : ',lrsam:3,'.');
         writeln(outfile,'Number of short range SAM''s     : ',srsam:3,'.');
         writeln(outfile,'Number of missile hits taken    : ',mhits:3,'.');
         writeln(outfile,'Number of bomb hits taken       : ',bhits:3,'.');
        end;          { with    }
      end;          { SHIPWRITE   }
```

```
begin
  reset (shipoutfile,'sba.2:shipoutfile.data');
  i := 0;
  page (outfile);
  writeln(outfile,'                    SHIPS    ');
  while NOT eof (shipoutfile) do
    begin
      i := i + 1;
      with shipoutfile^ do
        begin
          case class of
            cv   : kind := 'Carrier';
            dest : kind := 'Destroyer';
            crsr : kind := 'Cruiser';
          end;            {     case      }
          shipwrite;
          if i mod 3 = 0
            then begin
                   page (outfile);
                   writeln(outfile,'                    SHIPS  cont.    ');
                 end;
        end;              {      for/with     }
      get (shipoutfile);
    end;                  {      while     }
  close (shipoutfile,lock);
end;                      {    SHIPCHECK     }

PROCEDURE AIRCHECK;

  PROCEDURE ATTACKWRITE;

    begin
      with airoutfile^ do
       begin
        writeln(outfile);    writeln(outfile);    writeln(outfile);
        writeln(outfile,'      ',kind,' aircraft number ',num:3,'.');
        writeln(outfile);
        write  (outfile,'X-Y coordinate position             : (');
        writeln(outfile,xpos:4:1,',',ypos:4:1,').');
        writeln(outfile,'Current course / heading            : ',azmth:5,'.');
        writeln(outfile,'Current velocity                    : ',velcty:5,'.');
        writeln(outfile,'Current altitude                    : ',alt:5,'.');
        writeln(outfile,'Number of ASM''s                     : ',asm:5,'.');
        writeln(outfile,'Number of bomb''s                    : ',nbomb:5,'.');
      end;          {     with    }
    end;              {   ATTACKWRITE   }
```

```
PROCEDURE INTCPTWRITE;

  begin
    with airoutfile^ do
     begin
      writeln(outfile);    writeln(outfile);    writeln(outfile);
      writeln(outfile,'     ',kind,' number ',num:3,'.');
      writeln(outfile);
      write  (outfile,'X-Y coordinate position            : (');
      writeln(outfile,xpos:4:1,',',ypos:4:1,').');
      writeln(outfile,'Current course / heading           : ',azmth:5,'.');
      writeln(outfile,'Current velocity                   : ',velcty:5,'.');
      writeln(outfile,'Current altitude                   : ',alt:5,'.');
      writeln(outfile,'Number of AAM''s                    : ',aam:5,'.');
      writeln(outfile);
      writeln(outfile,'Note : Altitude = 0  ==> Aircraft is on carrier.');
    end;          {     with    }
  end;              {   ATTACKWRITE   }

PROCEDURE AEWWRITE;

  begin
    with airoutfile^ do
     begin
      writeln(outfile);    writeln(outfile);    writeln(outfile);
      writeln(outfile,'     ',kind,' aircraft number ',num:3,'.');
      writeln(outfile);
      write  (outfile,'X-Y coordinate position            : (');
      writeln(outfile,xpos:4:1,',',ypos:4:1,').');
      writeln(outfile,'Current course / heading           : ',azmth:5,'.');
      writeln(outfile,'Current velocity                   : ',velcty:5,'.');
      writeln(outfile,'Current altitude                   : ',alt:5,'.');
      writeln(outfile);
      writeln(outfile,'Note : Altitude = 0  ==> Aircraft is on carrier.');
    end;          {     with    }
  end;              {   ATTACKWRITE   }

begin               {      AIRCHECK     }
  reset (airoutfile,'aba.2:airoutfile.data');
  i := 0;
  page (outfile);
  writeln (outfile,'                AIRCRAFT          ');
  while NOT eof (airoutfile) do
    begin
      i := i + 1;
      with airoutfile^ do
        case iff of
          enemy : begin  kind := 'Enemy';  attackwrite;   end;
          friend : case acfrnd of
                     intcpt : begin
                                kind := 'Fighter/intercept';
                                intcptwrite;
                              end;
                     aew    : begin
```

```
                                    kind := 'Early warning';
                                    aewwrite;
                                  end;
                  end;             {     case   acfrnd   }
          end;                     {     case   iff      }
        get (airoutfile);
        if i mod 3 = 0
          then begin
                 page (outfile);
                 writeln (outfile,'              AIRCRAFT  cont.   ');
               end;
      end;            { while    }
    close (airoutfile,lock);
  end;                {       aircheck     }

begin                 {       CHECKFILES       }
  rewrite (outfile,'.rs232');
  shipcheck;
  aircheck;
  close (outfile,lock)
end.                  {       CHECKFILES       }
```

```
PROGRAM INTRO;

  USES chainstuff,thesistuff;

  PROCEDURE INTRODUCTION;

    begin
      screentro(clr);    screentro(off);   writeln;  writeln;
      writeln ('This is the AIR BATTLE ANALYSER (ABA) startup program.');
      writeln;
      writeln('If you have not already done so, please insert the ABA.2 disk ');
      writeln ('in disk drive number 2.');
      writeln;
      writeln ('Throughout the execution of this program you will be');
      writeln ('presented with several selection options.  In each instance');
      writeln ('please choose your desired option by typing the appropriate');
      writeln ('responce, if you here a beep, simply attempt to re-enter.');
      writeln;
      writeln ('Care should be taken not to hold down a selection key.    ');
      writeln ('It may produce undetermined results due to the auto repeat ');
      writeln ('function of the keyboard.                                 ');
      writeln;
      writeln('You may wish to have available some paper in order to make  ');
      writeln('notes to yourself, or to plot the current positions of the ');
      writeln('forces, and/or to determine movements of your forces.');
      writeln;   writeln;    continue;
    end;           {     introduction    }

  PROCEDURE INTROTWO;

    begin
      screentro(clr);    screentro(off);   writeln;
      writeln('For your convenience the initial positions are determined ');
      writeln('and layed out on a 100x100 grid where each unit is 10 NM. ');
      writeln('For example if the carrier is at (10,10) and an attack ');
      writeln('aircraft is at (89.5,10), the aircraft is 795 nautical miles ');
      writeln('due East of the carrier.  The grid is oriented as follows: ');
      writeln;
      writeln('        000  =  North is to the top.   ');
      writeln('        090  =  East  is to the right.  ');
      writeln('        180  =  South is to the bottom.  ');
      writeln('        270  =  West  is to the left.   ');
      writeln;
      writeln('Due to the nature of the screen and the computer graphics ');
      writeln('there will appear to be some distortions when the 100x100 ');
      writeln('grid is transformed to the screen''s dimensions.  Also, be');
      writeln('aware that the distances depicted on the graphic displays ');
      writeln('are decevtive due to the size of the figures.  ');
      writeln;      writeln;
      writeln ('                          Now Let''s Begin  ');
      writeln;     writeln;     continue;
    end;
```

```
PROCEDURE INTROTHREE;

  begin
    screentro(clr);    screentro(off);   writeln;   writeln;
    writeln('If you have played this game before then you may be familiar');
    writeln('with the default input data base and/or you may have ');
    writeln('previously remodeled it to suit your needs.');
    writeln;
    writeln('Please select one of the followins options :  ');
    writeln;     writeln;
    writeln('   1 :  BUILD DATABASE; WITH REVIEW of default database.');
    writeln;
    writeln('   2 :  BUILD DATABASE; WITHOUT REVIEW of default database.');
    writeln;
    writeln('   3 :  Use the [illegible] DATABASE parameters with NO REVIEW.  ');
    writeln;
    writeln('   4 :  Use the DATABASE parameters retained FROM LAST GAME.');
    writeln;      writeln;      writeln;
    write  ('  Type a number from 1 to 4  :   ');
    read (selection);
    screentro(on);
    while NOT (selection in ['1'..'4']) do
      begin
        screentro(bel);    writeln;
        write ('Must be a number from 1 to 4 !!  -- Please try again :  ');
        read (selection)
      end;
    end;


begin                                {     INTROTHESIS     }
  introduction;
  introtwo;
  introthree;
  case selection of
    '1' : begin
            setvar ('complete program');
            setchain ('.d2/thesis1.code');
          end;
    '2' : begin
            setvar ('allow change');
            setchain ('.d2/thesis1.code');
          end;
    '3' : begin
            setvar ('use default');
            setchain ('.d2/thesis1.code');
          end;
    '4' : begin
            setvar ('last parameters');
            setchain ('.d2/thesis3.code');
          end;
  end;          {   case   }
end.                                 {     INTROTHESIS     }
```

```
PROGRAM STARTEM;

  USES chainstuff,thesistuff;

  VAR
    shipinfile, shipdata : file of ship;
    airinfile, airdata : file of aircraft;
    kind,choice    : string;
    i,j      : 0..200;

  Procedure Parttwo;       forward;
  Procedure Partthree;     forward;
  Procedure Partfour;      forward;

  PROCEDURE DEFAULTFORCES;                {              PART ONE          }

    Begin
      screentro(clr);    screentro(off);   writeln;
      writeln;
      writeln ('                  FLEET COMPOSITION  ');
      writeln;
      writeln ('      1 Carrier - with;         ');
      writeln ('            2 squadrons of interceptors (24 aircraft).  ');
      writeln ('            6 AEW aircraft.  ');
      writeln ('            2 short range missile batteries (40 missiles).  ');
      writeln ('            Search radar and airborne intercept receivers.  ');
      writeln ('            Ship/ship and ship/air communications.  ');
      writeln;
      writeln ('      4 Cruisers (missile picket ships) - each with;  ');
      writeln ('            1 long range missile battery  (20 missiles).  ');
      writeln ('            1 short range missile battery (20 missiles).  ');
      writeln ('            Search radar and airborne intercept receivers.  ');
      writeln ('            Ship/ship and ship/air communications.  ');
      writeln;       parttwo;
    end;                                  {              PART ONE          }

  PROCEDURE PARTTWO;

    Begin
      writeln ('      4 Destroyers - each with;  ');
      writeln ('            1 short range missile battery (20 missiles).  ');
      writeln ('            Search radar and airborne intercept receivers.  ');
      writeln ('            Ship/ship and ship/air communications.  ');
      writeln;      continue;        partthree;
    end;                                  {              PARTTWO           }
```

```
PROCEDURE PARTTHREE;

  PROCEDURE MISSILEWRITE;

    begin
      screentro(clr);    screentro(off);   writeln;
      with shipinfile^ do
        begin
         writeln(' NOTE ::  DISTANCES  -are measured in- NAUTICAL MILES.   ');
         writeln;   writeln;
         writeln('                        MISSILES       ');
         writeln;
         writeln('    Short Range surface-to-air missiles     : SRSAM   ');
         writeln('       Probability of kill                  : ',srmpk:5:2);
         writeln('       Minimum range of firing envelope     : ',srmin:5);
         writeln('       Maximum range of firing envelope     : ',srmax:5);
         writeln;
         writeln('    Long Range surface-to-air missiles      : LRSAM  ');
         writeln('       Probability of kill                  : ',lrmpk:5:2);
         writeln('       Minimum range of firing envelope     : ',lrmin:5);
         writeln('       Maximum range of firing envelope     : ',lrmax:5);
         writeln;   writeln;
         writeln('                         RADARS       ');
         writeln;
         writeln('    Ship search radar maximum range         : ',ssrng:5);
         writeln('    Fire control radar maximum range        : ',fcrng:5);
         writeln;    continue;
        end;       {  with  }
    end;                              {          MISSILEWRITE          }

  PROCEDURE SHIPWRITE;

    begin
      with shipinfile^ do
        begin
          writeln;      writeln;
          writeln('Ship number ',i:2,' is a ',kind,'.');
          writeln;
          write  ('  X-Y coordinate position;  (x,y)      : ');
          writeln('(',xpos:4:1,',',ypos:4:1,')');
          writeln('  Direction of movement;    PIM         : ',pim:3);
          writeln('  Speed of advance;         SOA         : ',soa:3);
        end;   {  with  }
    end;                              {          SHIPWRITE             }

  begin                               {          PARTTHREE             }
    reset (shipinfile,'ABA.1:shipinfile.data');
    missilewrite;
    screentro(clr);    screentro(off);  writeln;
    writeln(' NOTE ::  SPEEDS -are measured in- KNOTS.  ');
    writeln;
    writeln('                   SHIP PARAMETERS          ');
    i := 0;
    while NOT eof (shipinfile) do
```

```
      begin
        i := i + 1;
        with shipinfile^ do
          begin
            case class of
              cv   : kind := 'carrier';
              dest : kind := 'destroyer';
              crsr : kind := 'cruiser';
            end;        {  case  }
            if ( i mod 2 = 1 ) and ( i <> 1 )
              then
                begin
                  writeln;  writeln;    writeln;
                  continue;                 screntro(clr);
                  screntro(off);          writeln;
                  writeln('                   SHIP PARAMETERS    cont. ');
                end;
            shipwrite;
          end;    {  with  }
        get (shipinfile);
      end;   {  while  }
    close (shipinfile,lock);
    writeln;       writeln;       continue;       partfour;
  end;                                {              PARTTHREE             }

PROCEDURE PARTFOUR;

  PROCEDURE AIRSETUP;

    begin
      screntro(clr);    screntro(off);  writeln;
      writeln('                         AIRCRAFT                ');
      writeln;
      writeln('The initial game set-up for the aircraft follows:');
      writeln;
      writeln('There is one CAP (combat air patrol) airborne consisting');
      writeln('of 2 fighter/intercept aircraft.  They are orbiting at');
      writeln('20,000 ft., approximately 5 NM ahead of the carrier ');
      writeln('All remaining fighters are onboard the carrier.');
      writeln;
      writeln('There are three AEW (airborne early warning) aircraft');
      writeln('deployed at 15,000 ft., about 200 NM ahead of the task force.');
      writeln('All other AEW aircraft are onboard the carrier.');
      writeln;
      writeln('All aircraft of the same type will carry the same ordnance');
      writeln('load and will have the same functional parameters.  ');
      writeln;
      writeln('Following is a sample parameter list for each aircraft type.');
      writeln;
      writeln;    writeln;    continue;
    end;
```

```
PROCEDURE AIRINFO  ;

  begin
   with airinfile^ do
    begin
      writeln;
      write  ('     X-Y coordinate position              : (');
      writeln(xpos:4:1,',',ypos:4:1,')');
      writeln('     Altitude                            : ',alt:6);
      writeln('     Course/heading                      : ',azmth:6);
      writeln('     Ground speed                        : ',velcty:6);
    end;   {  with  }
  end;                              {             AIRINFO             }

PROCEDURE ATTACKWRITE;

  begin
   with airinfile^ do
    begin
     screntro(clr);    screntro(off);  writeln;
      writeln('                    ENEMY AIRCRAFT                ');
      writeln;
      writeln('Attack aircraft number                  : ',i:2);
      writeln;
      writeln('  Number of air-to-surface missiles (ASM): ',asm:6);
      writeln('  ASM probability of kill               : ',asmpk:6:2);
      writeln('  Firing envelope (degrees about nose)  : ',asmenv:6);
      writeln('  ASM maximum range                     : ',asmrng:6);
      writeln('  Number of bombs                       : ',bomb:6);
      writeln('  Bomb probability of kill              : ',bombpk:6:2);
      airinfo;
      writeln;      writeln;        continue;
    end;          {     with          }
  end;                              {            ATTACKWRITE          }

PROCEDURE INTCPTWRITE;

  begin
   with airinfile^ do
    begin
     screntro(clr);    screntro(off);  writeln;
     writeln('                   FIGHTER/INTERCEPTOR          ');
     writeln;
     writeln('Aircraft endurance  (in minutes)         : ',intndr:6);
     writeln('Intercept radar detection envelope       : ',aienv:6);
     writeln('Intercept radar maximum range            : ',airng:6);
     writeln('Number of air-to-air missiles (AAM)      : ',aam:6);
     writeln('AAM probability of kill                  : ',aampk:6:2);
     writeln('AAM firing envelope (degrees about nose) : ',aamenv:6);
     writeln('AAM maximum range                        : ',aamrng:6);
     airinfo;
     writeln;      writeln;        continue;
    end;          {     with          }
  end;                              {          INTCPTWRITE           }
```

```
PROCEDURE AEWWRITE;

  begin
   with airinfile^ do
    begin
     screentro(clr);    screentro(off);  writeln;
     writeln('                    AIRBORNE EARLY WARNING       ');
     writeln;
     writeln('Aircraft endurance  (in minutes)          : ',aewndr:6);
     writeln('Air search radar maximum range            : ',aewrng:
     airinfo;
     writeln;       writeln;         continue;
    end;          {     with          }
  end;                                 {            ATTACKWRITE           }

begin                                  {             PARTFOUR             }
  reset (airinfile,'ABA.1:airinfile.data');
  airsetup;
  i := 0;
  while NOT eof (airinfile) do
    begin
      with airinfile^ do
        case iff of
          enemy  : begin
                     i :- i + 1;
                     attackwrite;
                   end;
          friend  : if alt > 0
                     then if acfrnd = aew
                            then aewwrite
                            else interwrite;
        end;                    {     case iff / with    }
      get (airinfile);
    end;                      {   while   }
  close (airinfile,lock);
end;                                   {            PARTFOUR              }

PROCEDURE GAMECHOISES;

  begin
    screentro(clr);    screentro(off);
    writeln;    writeln;   writeln;   writeln;
    writeln ('                How do you wish to set up the players? ');
    writeln;   writeln;
    writeln ('  1 :  Use the default fleet/ship and aircraft data-base.');
    writeln;
    writeln ('  2 :  Use the default fleet and build your own aircraft data.');
    writeln;
    writeln ('  3 :  Use the default aircraft and build your own fleet.');
    writeln;
    writeln ('  4 :  Build your own fleet and aircraft data-base.');
    writeln;  writeln;   writeln;   writeln;   writeln;
    write ('Type a number from 1 to 4 :    ');
```

```
    read   (selection);
    screentro(on);
    while NOT (selection in ['1'..'4']) do
      begin
        screentro(bel);
        writeln;
        write ('Must be a number from 1 to 4 !! -- Please try again:  ');
        read   (selection)
      end;
  end;                                          {          GAMECHOICES          }

PROCEDURE SHIPTRANSFER;

  begin
    reset   (shipinfile,'ABA.1:shipinfile.data');
    rewrite (shipdata  ,'ABA.2:shipdata.data');
    repeat
      shipdata^ := shipinfile^;
      put (shipdata);
      get (shipinfile);
    until eof (shipinfile);
    close (shipinfile,lock);
    close (shipdata  ,lock);
  end;

PROCEDURE AIRTRANSFER;

  begin
    reset   (airinfile ,'ABA.1:airinfile.data');
    rewrite (airdata   ,'ABA.2:airdata.data');
    repeat
      airdata^  :=  airinfile^;
      put ( airdata);
      get ( airinfile);
    until eof ( airinfile);
    close (airinfile ,lock);
    close (airdata   ,lock);
  end;

PROCEDURE PREPACK;

  begin
    screentro(clr);    screentro(off);
    writeln;    writeln;   writeln;
    writeln ('Since you have decided to use the default fleet and attack,');
    writeln ('you may wish to repeat the default database review. ');
    writeln;     writeln;    yesnosel;
    case selection of
      'Y','y' : begin
                  writeln;    writeln;    writeln;
                  continue;          defaultforces;
                end;
      'N','n' : begin
                  writeln;    writeln;    writeln;
```

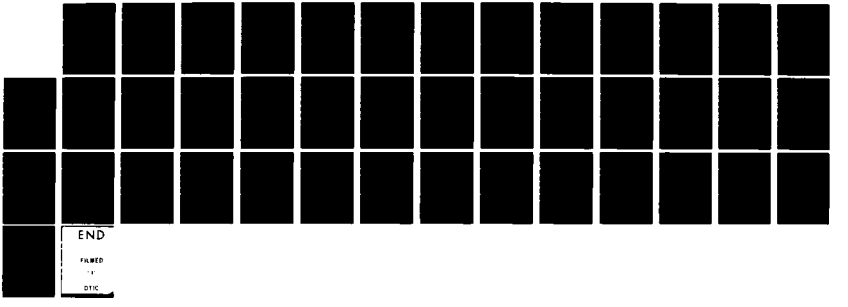
```
                continue;
              end;
    end;                    { case }
  end;

begin                                   { STARTEM }
  setval (choice);
  setchain ('.d2/thesis2.code');
  if choice = 'complete program'
    then  begin  defaultforces;   samechoices;  end
    else  if choice = 'allow change'
            then  begin  choice := 'no review';  samechoices;  end
            else  if choice = 'use default'
                    then  begin choice := 'no review'; selection := '1'; end
                    else  selection := '0';
  case selection of
    '1'  : begin
             if choice <> 'no review'    then    prepack;
             setcval ('default');
             shiptransfer;
             airtransfer;
           end;
    '2'  : begin
             setcval ('build air');
             shiptransfer;
           end;
    '3'  : begin
             setcval ('build fleet');
             airtransfer;
           end;
    '4'  : setcval ('build all');
  end     { case }
end.                                    { STARTEM }


PROGRAM CHANGEM;

  USES  chainstuff,thesistuff;

  VAR
    shipinfile, shipdata : file of ship;
    airinfile, airdata   : file of aircraft;
    aircomp  : aircraft;
    shipcomp : ship;
    choicekind : string;
    ijd        : 0..200;
    misschng   : boolean;
    length     : 0..10;
    temp       : integer;

  Procedure Redoship;     forward;
  Procedure Redomissiles; forward;
```

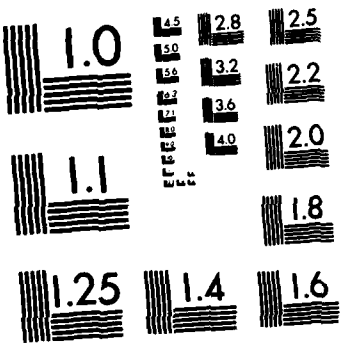
```
  Procedure Reducfriend;   forward;
  Procedure Reduenemy;     forward;
  Procedure Chngintcpt;    forward;
  Procedure Chngsaew;      forward;
  Procedure Coordinates;   forward;
  Procedure Alt_Azmth;     forward;
  Procedure Velocities;    forward;

  PROCEDURE CHECKVALUE ( lo, hi  : integer);

    begin
      temp := readint (length);
      while ((temp < lo)   or   (temp > hi)) do
        begin
          screentro(bel);    writeln;
          write  (' Please enter a value between ',lo:2,' and ',hi:3,' : ');
          temp:= readint(length);
        end;       {    while   }
    end;

  PROCEDURE NEWSCREEN;

    begin
      screentro(clr);    screentro(off);
      writeln;     writeln;    writeln;
    end;

{$include .d3/thesis2b.text}      {   A compiler instruction to include this   }
                                  {   text file when compiling the codefile.   }
{          Beginning text file Thesis2b.          }

  PROCEDURE COORDINATES;

    begin   with  airdata^ do    begin
      newscreen;
      writeln('The aircraft''s X-coordinate position            : ',xpos:4:1);
      if changeit
        then begin
               length := 4;
               xpos := readreal(length);
               writeln;
             end;
      screentro(off);   writeln;    writeln;    writeln;
      writeln('The aircraft''s Y-coordinate position            : ',ypos:4:1);
      if changeit
        then begin
               length := 4;
               ypos := readreal(length);
               writeln;
             end;
      end;       {   with   }
      writeln;   writeln;   continue;
    end;
```

```
PROCEDURE ALT_AZMTH;

  begin   with  airdata^ do    begin
    newscreen;
    writeln('The aircraft''s current altitude                  : ',alt:5);
    if changeit
      then begin
             length := 5;
             checkvalue (0,30000);
             alt := temp;
           end;
    screentro(off);   writeln;    writeln;    writeln;        '
    writeln('The aircraft''s current course/heading        : ',azmth:3);
    if changeit
      then begin
             length := 3;
             checkvalue (0,359);
             azmth := temp;
           end;
    end;        {   with   }
    writeln;    writeln;    continue;
  end;


PROCEDURE VELOCITIES;
  begin   with  airdata^ do    begin
    newscreen;
    writeln('The aircraft''s current ground speed (velocity) : ',velcty:4);
    if changeit
      then begin
             length := 4;
             checkvalue (0,2000);
             velcty := temp;
           end;
  end;        {   with  }
  end;

PROCEDURE REDOFRIEND;

  begin
    reset   (airinfile ,'ABA.1:airinfile.data');
    rewrite (airdata   ,'ABA.2:airdata.data');
    i := 0;    j := 0;
    repeat
      airdata^  :=  airinfile^;
      case airdata^.iff of
        enemy : put (airdata);
        friend : case airdata^.acfrnd of
                   intcpt :  begin    i := i + 1;    chngintcpt;    end;
                   aew    :  begin    j := j + 1;    chngaew    ;    end;
                 end;    {   case   }
      end;        {   case   }
      get ( airinfile);
```

AD-A124 638 AN INTERACTIVE MICROCOMPUTER WARGAME FOR AN AIR BATTLE (U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA J O WILSON OCT 82 2/2

UNCLASSIFIED F/G 15/7 NL





```
    until eof ( airinfile);
    close (airinfile ,lock);
    close (airdata   ,lock);
  end;                              {    REDOFRIEND  }

PROCEDURE CHNGINTCPT;

  Procedure Parttwo;        forward;
  Procedure Partthree;      forward;
  Procedure Partfour;       forward;

  PROCEDURE INTCPTPARA;             {     INTCPTPARA    partone   }

    begin  with  airdata^ do   begin
      screentro(off);    writeln;     writeln;     writeln;
      writeln('Endurance of aircraft (in minutes)            : ',intndr:3) ;
      if changeit
        then begin
               length := 3;
               checkvalue (0,255);
               intndr := temp;
             end;
      end;        {   with   }
      writeln;   writeln;   continue;     parttwo;
    end;

  PROCEDURE PARTTWO;                {     INTCPTPARA    parttwo   }

    begin   with  airdata^ do   begin
      newscreen;
      writeln('The number of air-to-air missiles   (AAM)       : ',aam:2);
      if changeit
        then begin
               length := 2;
               checkvalue (0,31);
               aam :- temp;
             end;
      screentro(off);     writeln;     writeln;     writeln;
      writeln('AAM probability of kill                         : ',aampk:4:2);
      if changeit
        then begin
               length := 4;
               aampk  := readreal(length);
               writeln;
             end;
      end;        {   with   }
      writeln;   writeln;   continue;     partthree;
    end;
```

```
PROCEDURE PARTTHREE;              {     INTCPTPARA     partthree     }

  begin   with  airdata^ do   begin
    newscreen;
    writeln('Radar detection envelope (degrees about nose)  : ',aienv:3);
    if changeit
      then begin
             length := 3;
             checkvalue (0,359);
             aienv := temp;
           end;
    screentro(off);     writeln;    writeln;    writeln;
    writeln('Intercept radar maximum range                  : ',airng:3);
    if changeit
      then begin
             length := 3;
             checkvalue (0,255);
             airng := temp;
           end;
    end;        {   with   }
    writeln;     writeln;      continue;        partfour;
                          {          INTCPTPARA     partthree          }

PROCEDURE PARTFOUR;               {     INTCPTPARA     partfour      }

  begin   with  airdata^ do   begin
    newscreen;
    writeln('AAM firing envelope (degrees about nose)       : ',aamenv:3);
    if changeit
      then begin
             length := 3;
             checkvalue (0,359);
             aienv := temp;
           end;
    screentro(off);     writeln;    writeln;    writeln;
    writeln('AAM maximum range                              : ',aamrng:3);
    if changeit
      then begin
             length := 3;
             checkvalue (0,255);
             aamrng := temp;
           end;
    end;        {   with   }
    writeln;     writeln;      continue;
  end;                        {          INTCPTPARA     partfour          }

begin                         {          CHNGINTCPT                       }
  newscreen;
  writeln('      Do you wish to change any of fighter/interceptor ');
  writeln('      aircraft number ',i:2,'''s parameters?');
  writeln;     yesnosel;     writeln;
  case selection of
    'Y','y' : begin
                coordinates;     alt_azmth;     velocities;
```

```
                intcptpara;      put (airdata);
              end;
    'N','n' : put (airdata);
  end;      { case }
end;                          {          CHNGINTCPT                    }

PROCEDURE CHNGAEW;

  PROCEDURE AEWPARA;

    begin  with airdata^ do begin
      screentro(off);     writeln;     writeln;     writeln;
      writeln('Endurance of aircraft (in minutes)              : ',aewndr:3);
      if changeit
        then begin
               length := 3;
               checkvalue (0,255);
               aewndr := temp;
             end;
      writeln;     writeln;      continue;
      newscreen;
      writeln('Airborne search radar maximum range             : ',aewrng:3);
      if changeit
        then begin
               length := 3;
               checkvalue (0,255);
               aewrng := temp;
             end;
      end;       { with }
      writeln;     writeln;      continue;
    end;                        {       AEWPARA      }

  begin
    newscreen;
    writeln('       Do you wish to change any of early warning ');
    writeln('       aircraft number ',j:2,'''s parameters?');
    writeln;     yesnosel;     writeln;
    case selection of
      'Y','y' : begin           {     CHNGAEW         }
                  coordinates;     alt_azmth;    velocities;
                  aewpara;      put (airdata);
                end;
      'N','n' : put (airdata);
    end;      { case }
  end;                          {     CHNGAEW         }
```

```
PROCEDURE MOREAIRINFO;

  begin
    with airdata^ do
      begin
        newscreen;       screncntro (on);
        length := 4;
        write  ('   Enter the X coordinate POSITION          : ');
        xpos := readreal(length);
        writeln;     writeln;      writeln;
        write  ('   Enter the Y coordinate POSITION          : ');
        ypos := readreal(length);
        writeln;     writeln;      writeln;
        length := 5;
        write  ('   Enter the ALTITUDE                       : ');
        checkvalue (0,30000);
        alt := temp;
        writeln;     writeln;      writeln;
        length := 3;
        write  ('   Enter the HEADING/COURSE                 : ');
        checkvalue (0,359);
        azmth := temp;
        writeln;     writeln;      writeln;
        length := 4;
        write  ('   Enter the VELOCITY                       : ');
        checkvalue (0,2000);
        velcty := temp;
        writeln;     continue;
      end;
  end;

PROCEDURE MASATTACK;

  begin
    with airdata^ do
      begin
        newscreen;       screncntro (on);
        length := 2;
        write  ('   Enter the number of ASM''s               : ');
        checkvalue (0,31);
        asm := temp;
        writeln;     writeln;      writeln;
        write  ('   Enter the number of bombs                : ');
        checkvalue (0,31);
        bomb := temp;
        writeln;     writeln;      writeln;    continue;
      end;       {     with   }
  end;
```

```
PROCEDURE MOREATTACK;

  begin
    with airdata^ do
      begin
        asmenv := aircomp.asmenv;
        asmrng := aircomp.asmrng;
        asmpk  := aircomp.asmpk ;
        bombpk := aircomp.bombpk ;
        moreairinfo;          masattack;
      end;          {    with    }
  end;

PROCEDURE REDOENEMY;

  Procedure Parttwo;          forward;
  Procedure Partthree;        forward;

  PROCEDURE ENEMYPARA;                {     ENEMYPARA   partone       }

    begin   with airdata^ do    begin
      newscreen;
      writeln('Number of air-to-surface missiles    (ASM)       : ',asm:2);
      if changeit
        then begin
               length := 2;
               checkvalue (0,31);
               asm := temp;
             end;
      screentro(off);   writeln;    writeln;    writeln;
      writeln('The ASM probability of kill                      : ',asmpk:4:2);
      if changeit
        then begin
               length := 4;
               asmpk  := readreal(length);
               writeln;
             end;
      end;          {   with    }
      writeln;   writeln;   continue;   parttwo;
    end;                            {    ENEMYPARA   partone     }

  PROCEDURE PARTTWO;                {     ENEMYPARA   parttwo       }

    begin   with airdata^ do    begin
      newscreen;
      writeln('ASM firing envelope (degrees about nose)     : ',asmenv:3);
      if changeit
        then begin
               length := 3;
               checkvalue (0,359);
               asmenv := temp;
             end;
      screentro(off);   writeln;    writeln;    writeln;
      writeln('ASM maximum range                            : ',asmrng:3);
```

```
    if changeit
      then begin
             length := 3;
             checkvalue (0,255);
             asmrns := temp;
           end;
    end;        {   with   }
    writeln;    writeln;    continue;    partthree;
  end;                      {    ENEMYPARA    parttwo    }

PROCEDURE PARTTHREE;        {    ENEMYPARA    partthree  }

  begin   with  airdata^ do    begin
    newscreen;
    writeln('The number of bombs                              : ',bomb:2);
    if changeit
      then begin
             length := 2;
             checkvalue (0,31);
             bomb := temp;
           end;
    screenctrl(off);   writeln;    writeln;    writeln;
    writeln('The probability of kill for each bomb            : ',bombpk:4:2);
    if changeit
      then begin
             length := 4;
             bombpk := readreal(length);
             writeln;
           end;
    end;        {   with   }
    writeln;      writeln;      continue;
  end;                      {    ENEMYPARA    partthree  }

begin                       {    REDOENEMY               }
  reset   (airinfile,'ABA.1:airinfile.data');
  rewrite (airdata   ,'ABA.2:airdata.data');
  i := 0;
  newscreen;
  writeln('     The attack aircraft are arranged in six formations of');
  writeln('     three aircraft.  You will be presented with each ');
  writeln('     aircraft in each formation.    ');
  writeln;      writeln;      writeln;
  repeat
    airdata^  :=  airinfile^;
    case airdata^.iff of
      friend : put (airdata);
      enemy : begin   i := i + 1;
                if i = 1
                  then   aircomp := airdata^
                  else   newscreen;
                write  ('   Do you wish to delete attack ');
                writeln('aircraft number ',i:2,'?');
                writeln;      yesnosel;      writeln;
                case selection of
```

```
                        'N','n' : begin
                                    screncntro(off); writeln; writeln; writeln;
                                    write  ('   Do you wish to change any of ');
                                    writeln('it''s parameters?');
                                    writeln;     yesnosel;     writeln;
                                    case selection of
                                      'Y','y' : begin
                                                  enemypara;
                                                  coordinates;     alt_azmth;
                                                  velocities;
                                                  writeln;      writeln;
                                                  continue;     put (airdata);
                                                end;
                                      'N','n' : begin
                                                  writeln;      writeln;
                                                  continue;     put (airdata);
                                                end;
                                    end;     { case  }
                                  end;
                      end;      { case  }
                    end;
        end;        { case    }
        get ( airinfile);
      until eof ( airinfile);
      repeat
        newscreen;
        writeln('   Do you wish to add more attack aircraft to the database?');
        writeln;    yesnosel;    writeln;
        if selection in ['Y','y'] then  begin  moreattack; put (airdata);  end;
      until selection in ['N','n'];
      close (airinfile ,lock);
      close (airdata   ,lock);
    end;                              {    REDOENEMY   }

{          End of text file Thesis2b.           }

  PROCEDURE FLEETBUILD;

    begin
      reset   (shipinfile,'ABA.1:shipinfile.data');
      rewrite (shipdata  ,'ABA.2:shipdata.data');
      newscreen;
      writeln ('  Do you wish to add or delete ships or ');
      writeln ('  change any of the ships'' parameters?');
      writeln;      writeln;     yesnosel;
      case selection of
        'Y','y' : redoship ;
        'N','n' : repeat
                    shipdata^ := shipinfile^;
                    put (shipdata);
                    get (shipinfile);
                  until eof (shipinfile);
      end;        {     case     }
      close (shipinfile,lock);
```

```
    close (shipdata  ,lock);
  end;

PROCEDURE AIRCRFTBUILD;

  begin
    newscreen;
    writeln ('    Do you wish to change/alter :');
    writeln;
    writeln('          1 :  Friendly aircraft parameters?  ');
    writeln;
    writeln('          2 :  Enemy aircraft parameters?     ');
    writeln;
    write             3 :  Both enemy and friendly aircraft parameters?  ');
    writeln;     writeln;
    write  ('  Type a number from 1 to 3  :   ');
    read (selection);
    screntro(on);
    while NOT (selection in ['1','2','3']) do
      begin
        screntro(bel);
        writeln;
        writeln ('    Must be a number from 1 to 3 !   ');
        write   ('          -- Please try again --   :    ');
        read (selection);
      end;       {    while     }
    case selection of
      '1' : redofriend;
      '2' : redoenemy;
      '3' : begin    redofriend;     redoenemy;    end;
    end;       {    case   }
  end;

PROCEDURE MOREINFO;

  begin
    with shipdata^ do
      begin
        newscreen;      screntro (on);
        length := 4;
        write  ('   Enter the X coordinate position        : ');
        xpos := readreal(length);
        writeln;    writeln;     writeln;
        write  ('   Enter the Y coordinate position        : ');
        ypos := readreal(length);
        writeln;    writeln;     writeln;
        length := 3;
        write  ('   Enter the PIM                          : ');
        checkvalue (0,359);
        pim := temp;
        writeln;     writeln;     writeln;
        length := 2;
        write  ('   Enter the SOA                          : ');
        checkvalue (0,30);
```

```
        sua := temp;
        writeln;      writeln;      writeln;     continue;
        newscreen;       screntro (on);
        length := 3;
        write  ('   Enter the surface search radar range   : ');
        checkvalue (0,255);
        ssrng := temp;
        writeln;      writeln;      writeln;
        write  ('   Enter the fire control radar range     : ');
        checkvalue (0,255);
        fcrng := temp;
        writeln;      writeln;      writeln;
        write  ('   Enter the number of Long Range SAM     : ');
        checkvalue (0,255);
        lrsam := temp;
        writeln;      writeln;      writeln;
        write  ('   Enter the number of Short Range SAM    : ');
        checkvalue (0,255);
        srsam := temp;
        writeln;      writeln;      writeln;     continue;
      end;       {     with   }
  end;

PROCEDURE MORESHIPS;

  begin
    with shipdata^ do
      begin
        lrmpk   := shipcomp.lrmpk ;
        srmpk   := shipcomp.srmpk ;
        lrmin   := shipcomp.lrmin ;
        lrmax   := shipcomp.lrmax ;
        srmin   := shipcomp.srmin ;
        srmax   := shipcomp.srmax ;
        mhits   := 0;
        bhits   := 0;
        newscreen;
        writeln('       Please choose a ship type : ');
        writeln;
        writeln('           1 :  Destroyer        ');
        writeln;
        writeln('           2 :  Cruiser          ');
        writeln;      writeln;
        write  ('       Type a number;  1 or 2 : ');
        read (selection);
        screntro(on);
        while not (selection in ['1','2']) do
          begin
            screntro(bel);     writeln;
            write  ('     Must be an available choice;  1 or 2 : ');
            read (selection);
          end;       {     while   }
        case selection of
          '1' : class := dest;
```

```
          '2' : class := crsr;
        end;          { case }
        moreinfo;
      end;          { with }
  end;

PROCEDURE REDOSHIP;

  Procedure Parttwo;          forward;
  Procedure Partthree;        forward;
  Procedure Partfour;         forward;

  PROCEDURE SHIPPARA;                    { SHIPPARA partone }

    begin  with shipdata^ do  begin
      newscreen;
      writeln('The ship''s X-coordinate position          : ',xpos:4:1);
      if changeit
        then begin
               length := 4;
               xpos := readreal(length);
               writeln;
             end;
      screentro(off);  writeln;  writeln;  writeln;
      writeln('The ship''s Y-coordinate position          : ',ypos:4:1);
      if changeit
        then begin
               length := 4;
               ypos := readreal(length);
               writeln;
             end;
      end;          { with }
      writeln;  writeln;  continue;  parttwo;
    end;                    { SHIPPARA partone }

  PROCEDURE PARTTWO;        { SHIPPARA parttwo }

    begin  with shipdata^ do  begin
      newscreen;
      writeln('The ship''s PIM (position of intended movement)  : ',pim:3);
      if changeit
        then begin
               length := 3;
               checkvalue (0,359);
               pim := temp;
             end;
      screentro(off);  writeln;  writeln;  writeln;
      writeln('The ship''s actual SOA (speed of advance)    : ',soa:2);
      if changeit
        then begin
               length := 2;
               checkvalue (0,50);
               soa := temp;
             end;
```

```
    end;        { with }
    writeln;    writeln;    continue;    partthree;
  end;                      {     SHIPPARA    parttwo     }

PROCEDURE PARTTHREE;        {     SHIPPARA    partthree   }

  begin   with shipdata^ do    begin
    newscreen;
    writeln('Surface search radar maximum range              : ',ssrng:3);
    if changeit
      then begin
             length := 3;
             checkvalue (0,255);
             ssrng := temp;
           end;
    screentro(off);   writeln;    writeln;    writeln;
    writeln('Fire control radar maximum range                : ',fcrng:3);
    if changeit
      then begin
             length := 3;
             checkvalue (0,255);
             fcrng := temp;
           end;
    end;        { with }
    writeln;    writeln;    continue;       partfour;
  end;                      {     SHIPPARA    partthree   }

PROCEDURE PARTFOUR ;        {     SHIPPARA    partfour    }

  begin   wi'  shipdata^ do    begin
    newscreen;
    writeln('Number of long range surface-to-air missiles    : ',lrsam:3);
    if changeit
      then begin
             length := 3;
             checkvalue (0,255);
             lrsam := temp;
           end;
    screentro(off);   writeln;    writeln;    writeln;
    writeln('Number of short range surface-to-air missiles   : ',srsam:3);
    if changeit
      then begin
             length := 3;
             checkvalue (0,255);
             srsam := temp;
           end;
    end;        { with }
    writeln;    writeln;    continue;
  end;                      {     SHIPPARA    partfour    }

begin                       {     REDOSHIP                }
  i := 0;
  repeat
    i := i + 1;
```

```
    shipdata^ := shipinfile^;
    shipcomp := shipinfile^;
    if i = 1  then       redomissiles;
    if misschng
      then  with shipdata^ do
              begin
                lrmpk  := shipcomp.lrmpk ;
                srmpk  := shipcomp.srmpk ;
                lrmin  := shipcomp.lrmin ;
                lrmax  := shipcomp.lrmax ;
                srmin  := shipcomp.srmin ;
                srmax  := shipcomp.srmax ;
              end;          {    with/if   }
    case shipdata^.class of
      cv   : kind := 'carrier';
      dest : kind := 'destroyer';
      crsr : kind := 'cruiser';
    end;
    newscreen;
    if shipdata^.class <> cv
      then
        begin
          write  ('       Do you want to delete ship number ');
          writeln(i:2,' a ',kind,'?');
          writeln;       yesnosel;
          case selection of
            'N','n' : begin
                        screncntro(off);   writeln;  writeln;  writeln;
                        write  ('       Do you wish to change ');
                        writeln('any of the parameters?');
                        writeln;       yesnosel;
                        case selection of
                          'Y','y' : begin  shippara;   put (shipdata);  end;
                          'N','n' : put (shipdata);
                        end;         {   case   }
                      end;
          end;               {     case     }
        end
      else
        begin
          writeln('       Do you wish to change any of the parameters ');
          writeln('       for ship number ',i:2,', a ',kind,'?     ');
          writeln;       yesnosel;
          case selection of
            'Y','y' : begin  shippara;   put (shipdata);  end;
                        writeln;         writeln;
                        continue;        put (shipdata);
                      end;
          end;        {   case   }
        end;          {    if    }
    get (shipinfile);
  until eof (shipinfile);
  repeat
```

```
      newscreen;
      writeln('       Do you wish to add more ships to the database? ');
      writeln;     yesnosel;
      if selection in ['Y','y'] then  begin moreships;  put (shipdata);  end;
    until selection in ['N','n']
  end;                              {    REDOSHIP    }

PROCEDURE REDOMISSILES;

  Procedure Parttwo;       forward;
  Procedure Partthree;     forward;

  PROCEDURE MISSPARA;        {     MISSPARA   partone    }

    begin   with shipcomp  do    begin
      newscreen;
      writeln('Long range SAM probability of kill          : ',lrmpk:4:2);
      if changeit
        then begin
               length := 4;
               lrmpk  := readreal(length);
               writeln;
             end;
      screentro(off);   writeln;    writeln;    writeln;
      writeln('Short range SAM probablity  of kill         : ',srmpk:4:2);
      if changeit
        then begin
               length := 4;
               srmpk  := readreal(length);
               writeln;
             end;
      end;        {   with   }
      writeln;       writeln;      continue;     parttwo;
    end;                        {     MISSPARA   partone    }
  PROCEDURE PARTTWO;            {     MISSPARA   parttwo    }

    begin   with shipcomp  do    begin
      newscreen;
      writeln('Short range missile minimum target distance     : ',srmin:3);
      if changeit
        then begin
               length := 3;
               checkvalue (0,255);
               srmin := temp;
             end;
      screentro(off);   writeln;    writeln;    writeln;
      writeln('Short range missile maximum target distance      : ',srmax:3);
      if changeit
        then begin
               length := 3;
               checkvalue (0,255);
               srmax := temp;
             end;
```

```
      end;        { with }
      writeln;    writeln;     continue;     partthree;
    end;                      { MISSPARA   parttwo    }

  PROCEDURE PARTTHREE;        { MISSPARA   partthree  }

    begin  with shipcomp do   begin
      newscreen;
      writeln('Long range missile minimum target distance      : ',lrmin:3);
      if changeit
        then begin
               length := 3;
               checkvalue (0,255);
               lrmin := temp;
             end;
      screentro(off);  writeln;   writeln;   writeln;
      writeln('Long range missile maximum target distance      : ',lrmax:3);
      if changeit
        then begin
               length := 3;
               checkvalue (0,255);
               lrmax := temp;
             end;
      end;        { with }
      writeln;    writeln;     continue;
    end;                      { MISSPARA   partthree  }

  begin                       { REDOMISSILES          }
    newscreen;
    writeln('      Do you wish to change any of the ships''    ');
    writeln('      surface-to-air missile parameters?       ');
    writeln;
    writeln('      These parameters will be equivalent for all ships.');
    writeln;   writeln;     yesnosel;
    case selection of
      'Y','y' : begin  missparm;  misschng := true;    end;
      'N','n' : misschng := false;
    end;        { case }
  end;                        { REDOMISSILES  }

begin                         { CHANGEM       }
  setchain ('.d2/thesis3.code');
  getval (choice);
  if choice = 'build all'
    then    begin    fleetbuild;    aircraftbuild;     end
    else  if choice = 'build fleet'
            then  fleetbuild
            else  if choice = 'build air'
                    then  aircraftbuild;
end.                          { CHANGEM     }
```

```
{$swap+}              { A compiler option for decreasing memory usage.      }
PROGRAM ABAGAME;

  USES
    pgraf,realmodes,transcend,applestuff,thesistuff,
    {$using .d2/thesis3.lib} makeforms,grafstuff,bearings;

  CONST
    incr = 20;        { The interval that the attack updates it's headins.  }
    retrtime = 120;   { Time of retreat for the attack.                     }
    retrthds = 90;    { Retreat headins for the attack.                     }
    retrtalt = 20000;{ Retreat altitude for the attack.                     }
    retrtvel = 400;   { Retreat velocity for the attack.                    }
    inbdist = 200;    { Distance from carrier to descend for attack.        }
    inbualt = 200;    { Inbound altitude for attack.                        }
    tol = 0.5;        { Tolerance for bombing position comparisons.         }
    hittol = 7;       { Tolerance number of hits a ship can take .          }
    recov =  5;       { Radial distance around carrier for aircraft recovery.}
    tmdefault = 160;

  VAR
    shipdata,shipoutfile     : file of ship;
    airdata,airoutfile       : file of aircraft;
    entrone, entrtwo       : shipntr;
    shipbase, shipnext     : shipntr;
    airbase,  airnext      : airentr;
    atkbase,  atknext      : airentr;
    cntbase,  cntnext      : cntentr;
    cvx,cvy : real;
    choice,kind,msg : string;
    frcnum,i,j,len   : 0..255;
    stop,stepthru    : boolean;
    mx,my,scale,look,time,endtime,tmster     : integer;
    players,frmplay,enmplay,newcenter     : set of 0..511;

  PROCEDURE SHOWFLEET  ;                                forward;
  PROCEDURE SHOWATTACK ;                                forward;
  PROCEDURE SHIPSTATUS ;                                forward;
  PROCEDURE INTSTATUS  ;                                forward;
  PROCEDURE AEWSTATUS  ;                                forward;
  PROCEDURE RDRSTATUS  ;                                forward;
  PROCEDURE AIRDELETE  ( thisun : airentr;
                         VAR base : airentr );          forward;
  PROCEDURE SHIPDELETE ( thisun : shipntr;
                         VAR base : shipntr );          forward;

{$include /progfiles/thesis3b.text}
                                { A compiler instruction to include this    }
                                { text file when compiling the codefile.    }
```

```
        {         Beginning text file Thesis3b.       }

SEGMENT PROCEDURE SHOWFORMS;

  begin
    grafixmode (bw280,1);
    grafixon;     misforms;    airforms;
    mss := 'Presented below are the figures that     ';
    moveto (0,190);      unitwrite (3,mss[1],length(mss),0,12);
    mss := 'will be used in the graphic displays.    ';
    moveto (0,180);      unitwrite (3,mss[1],length(mss),0,12);
    mss := 'The actual position of the unit shown    ';
    moveto (0,160);      unitwrite (3,mss[1],length(mss),0,12);
    mss := 'will be the upper left point of figure.  ';
    moveto (0,150);      unitwrite (3,mss[1],length(mss),0,12);
    mss := '       SHIPS              AIRCRAFT       ';
    moveto (0,120);      unitwrite (3,mss[1],length(mss),0,12);
    mss :=                    FIGHTER     AEW        ';
    moveto (0,110);      unitwrite (3,mss[1],length(mss),0,12);
    moveto (59 ,100);     drawimage (shipform,2,0,0,8,6);     dotat (59 ,100);
    moveto (152,100);     drawimage (intform ,2,0,0,10,6);    dotat (152,100);
    moveto (225,100);     drawimage (aewform ,2,0,0,10,6);    dotat (225,100);
    mss :=                RADAR CONTACTS                 ';
    moveto (0,70);         unitwrite (3,mss[1],length(mss),0,12);
    mss := ' AEW  FIRE CNTRL  AIR INTCPT  SHIP SRCH ';
    moveto (0, 50);          unitwrite (3,mss[1],length(mss),0,12);
    moveto (14 ,40);     drawimage (aewrdr ,2,0,0,5,5);
    moveto (70 ,40);     drawimage (fcrdr  ,2,0,0,5,5);
    moveto (134,40);     drawimage (airdr  ,2,0,0,5,5);
    moveto (238,40);     drawimage (ssrdr  ,2,0,0,5,5);
    mss := 'Press SPACE BAR to continue';
    moveto (40,8);         unitwrite (3,mss[1],length(mss),0,12);
    read (selection);
  end;                    {        SHOWFORMS       }

SEGMENT PROCEDURE POSITRANSFER ;
  begin
    fltbase := nil;
    shipnext := shipbase;
    fromplay := [];
    while shipnext <> nil  do
      begin
        new (fltnext);
        fltnext^.xpos := shipnext^.xpos;
        fltnext^.ypos := shipnext^.ypos;
        fltnext^.num  := shipnext^.num ;
        fromplay := fromplay + [shipnext^.num];
        fltnext^.what := boat;
        fltnext^.link := fltbase;
        fltbase := fltnext;
        shipnext := shipnext^.link;
      end;          {     while     }
    airnext := airbase;
```

```
while airnext <> nil  do
  begin
    if ((airnext^.xpos <> cvx)   or   (airnext^.ypos <> cvy))
      then  begin
              new (fltnext);
              fltnext^.xpos := airnext^.xpos;
              fltnext^.ypos := airnext^.ypos;
              fltnext^.num  := airnext^.num ;
              frnplay := frnplay + [airnext^.num];
              if airnext^.acfrnd = aew
                then  fltnext^.what := early
                else  fltnext^.what := fight;
              fltnext^.link := fltbase;
              fltbase := fltnext;
            end;     {    if    }
    airnext := airnext^.link;
  end;       {    while   }
cntnext := cntbase;
while cntnext <> nil  do
  begin
    new (fltnext);
    fltnext^.xpos := cntnext^.xpos;
    fltnext^.ypos := cntnext^.ypos;
    fltnext^.num  := cntnext^.num;
    frnplay := frnplay + [cntnext^.num];
    case cntnext^.who of
      asrch : fltnext^.what := aewcnt;
      ssrch : fltnext^.what := sscnt ;
      ai    : fltnext^.what := aicnt ;
      fcon  : fltnext^.what := fccnt ;
    end;         {    case    }
    fltnext^.link := fltbase;
    fltbase := fltnext;
    cntnext := cntnext^.link;
  end;
enebase := nil;
atknext := atkbase;
eneplay := [];
while atknext <> nil  do
  begin
    new (enenext);
    enenext^.xpos := atknext^.xpos;
    enenext^.ypos := atknext^.ypos;
    enenext^.num  := atknext^.num ;
    eneplay := eneplay + [atknext^.num];
    enenext^.what := mine;
    enenext^.link := enebase;
    enebase := enenext;
    atknext := atknext^.link;
  end;
end;
```

```
SEGMENT PROCEDURE WHATNEXT;

  begin
    msg := '  U(pscale  /  D(ownscale  /  R(ecenter ';
    moveto (0,191);       unitwrite(3,msg[1],length(msg),0,12);
    dotat (0,11);      dotat (0,13);
    moveto (0,12);     lineto (scale,12);
    dotat (scale,11);      dotat (scale,13);     moverel (2,3);
    msg := ' ==> 10 NM';       unitwrite(3,msg[1],length(msg),0,12);
    msg := ' C(ontinue                Time : ';
    moveto (0,8);       unitwrite(3,msg[1],length(msg),0,12);
    str (time,msg);
    moveto (220,8);       unitwrite(3,msg[1],length(msg),0,12);
    grafixon;        read (selection);
    while not (selection in ['C','c','U','u','D','d','R','r']) do
      begin
        screentro(bel);      read (selection);
      end;
    case selection of
      'U','u' :  begin      scaleup;        scale := scale div 2;       end;
      'D','d' :  begin      scaledown;      scale := scale * 2;         end;
      'R','r' :  begin
                   screentro(clr);      screentro(off);    texton;
                   writeln;      writeln;      writeln;      writeln;
                   writeln('  Choose a unit number from the last display ');
                   writeln('  on which you wish to recenter the display. ');
                   writeln;      writeln;
                   if atksraf
                     then     players := eneplay
                     else     players := frnplay;
                   write  ('  Must be a DISPLAYED number  :  ');
                   screentro(on);
                   len := 3;      nucent := readint(len);
                   while  ((nucent < 0)   or   (nucent > 511))  do
                     begin
                       screentro(bel);       writeln;
                       write  ('  Must be a DISPLAYED number  :  ');
                       nucent := readint(len);
                     end;      {    while    }
                   newcenter := [nucent];
                   while   not (newcenter <= players)  do
                     begin
                       screentro(bel);       writeln;
                       write  ('  Must be a DISPLAYED number  :  ');
                       nucent := readint(len);
                       while  ((nucent < 0)   or   (nucent > 511))  do
                         begin
                           screentro(bel);       writeln;
                           write  ('  Must be a DISPLAYED number  :  ');
                           nucent := readint(len);
                         end;      {    while    }
                       newcenter := [nucent];
                     end;      {    while    }
                   recenter;
```

```
                end;
    end;      {     case    }
  end;

SEGMENT PROCEDURE ACFTMOVES;

  Var
    hdg,vel,altd : integer;

  begin
    write  (' Enter desired heading of aircraft  : ');
    len := 3;     hdg := readint(len);   writeln;  writeln;
    while (hdg > 359) do
      begin
        writeln(' Heading must be between 000 and 359. ');  writeln;
        write  (' Enter desired heading : ');
        hdg := readint(len);   writeln;   writeln;
      end;      {   while   }
    airnext^.azmth := hdg;
    write  (' Enter desired velocity of aircraft : ');
    len := 4;     vel := readint(len);   writeln;  writeln;
    while (vel > 2000) do
      begin
        screntro(bel);
        writeln(' Velocity must be between 0 and 2000.  ');   writeln;
        write  (' Enter desired velocity : ');
        vel := readint(len);   writeln;   writeln;
      end;      {   while   }
    airnext^.velcty := vel;
    write  (' Enter desired altitude of aircraft (MUST be < 25000) : ');
    len := 5;     altd := readint(len);   writeln;  writeln;
    while (altd > 25000) do
      begin
        screntro(bel);
        writeln(' Altitude MUST be between 0 and 25000. ');    writeln;
        write  (' Enter desired altitude : ');
        altd := readint(len);   writeln;   writeln;
      end;      {   while   }
    airnext^.alt := altd;
  end;

SEGMENT PROCEDURE MOVEAC ( whatac : frndtype;
                           dowhat : string   );
  Var
    des,dis,u : integer;
    dowhatnum : 1..10;

  begin
    airment := airbase;     u := 1;
    if dowhat = 'launch'
      then dowhatnum := 1
      else if dowhat = 'move'
```

```
          then dowhatnum := 2
          else dowhatnum := 3;
screentro(clr);   screentro(off);   writeln;
while airnext <> nil do
  begin
   screentro(clr);   screentro(off);   writeln;
   if j = 1 then  begin
    case whatac of
      intcpt :  kind := 'fighter';
      aew    :  kind := 'early warning aircraft';
    end;    {  case  }
    case dowhatnum of
      1   : if   ((airnext^.alt = 0)    and    (airnext^.acfrnd = whatac)
              and   (airnext^.xpos = cvx)  and  (airnext^.ypos = cvy))
               then begin
                      if whatac = aew
                        then airnext^.aewndr := 240
                        else begin
                               airnext^.intndr := 120;
                               airnext^.aam := 6;
                             end;
                      writeln;   writeln;
                      writeln(' Launching a ',kind,'!');
                      writeln;  writeln;
                      acftmoves;
                      j := j + 1;
                    end;
     2,3 : if ((airnext^.alt <> 0) and (airnext^.acfrnd = whatac))
             then begin
                  des := degrees  (cvx,cvy,airnext^.xpos,airnext^.ypos);
                  dis := distance (cvx,cvy,airnext^.xpos,airnext^.ypos);
                  dis := dis * 10;
                  write  ('  Do you wish to ',dowhat,' the ',kind);
                  writeln(', number ',airnext^.num:2,' located ');
                  write  ('  ',des:3,' degrees, ',dis:3);
                  writeln(' NM from carrier? ');
                  if dowhat = 'move'
                    then begin
                           writeln;   writeln;
                           write  ('     Current heading is   : ');
                           writeln(airnext^.azmth:5);   writeln;
                           write  ('     Current velocity is  : ');
                           writeln(airnext^.velcty:5);   writeln;
                           write  ('     Current altitude is  : ');
                           write  (airnext^.alt:5);
                           if (airnext^.alt = 1)
                             then begin
                                    write  ('          ');
                                    writeln('Aircraft is being RECOVERED');
                                    writeln;
                                  end
                             else begin   writeln;     writeln;   end;
                         end;
                  writeln;    yesnosel;   writeln;   writeln;
```

```
                        if selection in ['Y','y']
                          then  if dowhat = 'move'
                                  then begin acftmoves;   j := j + 1;   end
                                  else begin
                                         airnext^.alt    := 1;
                                         airnext^.velcty := 350;
                                         if des < 180
                                           then  airnext^.azmth := des + 180
                                           else  airnext^.azmth := des - 180;
                                         j := j + 1;
                                       end;      {   if yes   }
                    end;         {  if case 2,3      }
          end;     {  case   }
        end;     {   if j = 1  }
        airnext := airnext^.link;
      end;     {   while   }
  end;

SEGMENT PROCEDURE SHIPMOVES;
    crs,spd : integer;

  begin
    write  ('  Enter desired PIM (course)          : ');
    len := 3;      crs := readint(len);   writeln;  writeln;
    while (crs > 359) do
      begin
        screentro(bel);
        writeln('   Course must be between 000 and 359.');
        writeln;
        write  ('   Enter desired PIM      : ');
        crs := readint(len);   writeln;   writeln;
      end;       {   while   }
    shipnext^.pim := crs;
    write  ('  Enter desired speed of advance      : ');
    len := 2;      spd := readint(len);   writeln;  writeln;
    while (spd > 30) do
      begin
        screentro(bel);
        writeln('   Speed must be between 0 and 30.');
        writeln;
        write  ('   Enter desired speed     : ');
        spd := readint(len);   writeln;   writeln;
      end;       {   while   }
    shipnext^.spd := spd;
  end;
```

```
SEGMENT PROCEDURE MOVESHIP (dowhat : string);

  VAR
    des,dis,j : integer;
    npim      : 0..359;
    nsoa      : 0..30;
    dowhatnum : 1..10;

  begin
    shipnext := shipbase;     j := 1;
    if dowhat = 'single'
      then dowhatnum := 1
      else dowhatnum := 2;
    while shipnext <> nil do
      begin
       screntro(clr);   screntro(off);   writeln;
       if j = 1 then begin
        case dowhatnum of
          1 : begin
                case shipnext^.class of
                  cv   :  kind := 'carrier';
                  dest :  kind := 'destroyer';
                  crsr :  kind := 'cruiser';
                end;       { case   }
                des := degrees(cvx,cvy,shipnext^.xpos,shipnext^.ypos);
                dis := distance(cvx,cvy,shipnext^.xpos,shipnext^.ypos) * 10;
                write  ('  Do you wish to move the ',kind,' number ');
                write  (shipnext^.num:2);
                if shipnext^.class = cv
                  then  writeln('?')
                  else  begin
                          writeln(' located ');
                          write  ('  ',des:3,' degrees, ',dis:3);
                          writeln(' NM from carrier? ');
                        end;     { if  }
                writeln;   writeln;
                writeln('       Current PIM : ',shipnext^.pim);
                writeln;
                writeln('       Current SOA : ',shipnext^.soa);
                writeln;    yesnosel;   writeln;   writeln;
                if selection in ['Y','y']
                  then begin   shipmoves;     j := j + 1;    end;
              end;
          2 : begin
                writeln;   writeln;
                writeln('  Altering the course / speed of the fleet : ');
                writeln;   writeln;
                writeln('       Current PIM : ',shipnext^.pim);
                writeln;
                writeln('       Current SOA : ',shipnext^.soa);
                shipmoves;
                npim := shipnext^.pim;
                nsoa := shipnext^.soa;
                shipnext := shipnext^.link;
```

```
                repeat
                  shipnext^.pim := npim;
                  shipnext^.soa := nsoa;
                  shipnext := shipnext^.link;
                until shipnext = nil;
              end;
        end;      {   case    }
       end;       {   if j = 1   }
       if dowhat <> 'fleet'  then shipnext := shipnext^.link;
      end;     {    while  }
  end;

SEGMENT PROCEDURE MAKECNTC ( whatrdr   : radartype;
                             deadflag  : boolean;
                             x,y       : real;
                             nmb1,nmb2 : integer   );

  VAR
    newcntc : boolean;
    ix,iy,cx,cy   : integer;

  begin
    cntnext := cntbase;
    newcntc := true;
    ix := round (x);    iy := round (y);
    while cntnext <> nil do
      begin
        cx := round (cntnext^.xpos);    cy := round (cntnext^.ypos);
        if ((cx = ix)  and  (cy = iy))
          then if cntnext^.dead
                 then  newcntc := false
                 else if deadflag
                        then begin
                               newcntc := false;
                               cntnext^.who  := whatrdr;
                               cntnext^.rdnm := nmb2;
                               cntnext^.dead := deadflag;
                             end
                        else if ord(whatrdr) > ord(cntnext^.who)
                               then begin
                                      newcntc := false;
                                      cntnext^.who := whatrdr;
                                      cntnext^.rdnm := nmb2;
                                    end
                               else  newcntc := false;
        cntnext := cntnext^.link;
      end;
    if newcntc
      then begin
             new (cntnext);
             cntnext^.xpos := x;
             cntnext^.ypos := y;
             cntnext^.num  := frcnum + nmb1;
             cntnext^.who  := whatrdr;
```

```
            cntnext^.rdnm := nmb2;
            cntnext^.dead := deadflag;
            cntnext^.link := cntbase;
            cntbase := cntnext;
          end;
  end;

SEGMENT PROCEDURE SHIPRADARCNTC;

  VAR
    dist,brg,lt,rt : integer;
    ax,ay,rh       : real;
    nm             : 0..255;

  begin
    randomize;
    shipnext := shipbase;
    while shipnext <> nil do
      with shipnext^ do
        begin
          atknext := atkbase;
          scrncntrc(lf);    write  ('=> ');
          while atknext <> nil do
            begin
              ax := atknext^.xpos;      ay := atknext^.ypos;
              nm := atknext^.num;
              rh := [illegible] .alt);
              dist := distance(xpos,ypos,ax,ay) * 10;
              brg  := degrees (ax,ay,xpos,ypos);
              lt := (atknext^.azmth - atknext^.asmenv div 2);
              rt := (atknext^.azmth + atknext^.asmenv div 2);
              if (lt <   0)  then  lt := lt + 360;
              if (rt > 360)  then  rt := rt - 360;
              if ((dist <= srmax)    and   (dist > srmin)
                   and (srsam > 0)   and   (dist <= rh) )
                then begin
                       srsam := srsam - 1;
                       if random <= maxint * srmpk
                         then   makecntc (fcon,true,ax,ay,nm,num)
                         else   makecntc (fcon,false,ax,ay,nm,num);
                     end
                else if ( (dist <= lrmax)   and   (dist > lrmin)
                           and   (lrsam > 0)   and   (dist <= rh) )
                       then begin
                              lrsam := lrsam - 1;
                              if random <= maxint * lrmpk
                                then   makecntc (fcon,true,ax,ay,nm,num)
                                else   makecntc (fcon,false,ax,ay,nm,num);
                            end
                       else if ((dist <= fcrng)   and   (dist <= rh))
                              then  makecntc (fcon,false,ax,ay,nm,num)
                              else if ((dist <= ssrng)  and  (dist <= rh))
                                     then makecntc (ssrch,false,ax,ay,nm,num);
              if ((dist <= tol)  and  (atknext^.bomb > 0))
```

```
                then begin
                       atknext^.bomb := atknext^.bomb - 1;
                       if random <= maxint * atknext^.bombpk
                         then    bhits := bhits + 1;
                     end
                else if ((dist <= atknext^.asmrng)  and  (atknext^.asm > 0)
                          and  (brg >= lt)   and   (brg <= rt))
                       then begin
                              atknext^.asm := atknext^.asm - 1;
                              if random <= maxint * atknext^.asmpk
                                then   mhits := mhits + 1;
                            end;
              if ((bhits + mhits) > hittol)
                then  sunk := true;
              atknext := atknext^.link;
            end;       {     while   atknext     }
          shipnext := link;
        end;           {  with    }
  end;

SEGMENT PROCEDURE AIRADARCNTC;
  VAR
    dist,brg,ltr,rtr,ltm,rtm  : integer;
    ax,ay,rh    : real;
    nm      : 0..255;

  begin
    randomize;
    airnext := airbase;
    scrncntro(clr);     scrncntro(on);     texton;
    while airnext <> nil do
      begin
        if airnext^.alt > 0
          then with airnext^ do
                 begin
                   atknext := atkbase;
                   scrncntro(lf);      write  ('=> ');
                   while atknext <> nil do
                     begin
                       ax := atknext^.xpos;     ay := atknext^.ypos;
                       nm := atknext^.num;
                       rh := 1.25 * (sqrt(atknext^.alt) + sqrt(alt));
                       dist := distance(xpos,ypos,ax,ay) * 10;
                       brg  := degrees (xpos,ypos,ax,ay);
                       ltr := (azmth - aienv  div 2);
                       rtr := (azmth + aienv  div 2);
                       if (ltr <    0)  then  ltr := ltr + 360;
                       if (rtr > 360)  then  rtr := rtr - 360;
                       ltm := (azmth - aamenv div 2);
                       rtm := (azmth + aamenv div 2);
                       if (ltm <    0)  then  ltm := ltm + 360;
                       if (rtm > 360)  then  rtm := rtm - 360;
                       case acfrnd of
```

```
                        aew    : if ((dist <= aewrng)  and  (dist <= rh))
                                   then   makecntc (asrch,false,ax,ay,nm,num);
                        intcpt : if ((dist <= aamrng)  and  (aam > 0)
                                   and   (brg >= ltm)  and  (brg <= rtm)
                                   and   (dist <= rh))
                                   then begin
                                          aam := aam - 1;
                                          if random <= maxint * aampk
                                           then makecntc (ai,true,ax,ay,nm,num)
                                           else makecntc (ai,false,ax,ay,nm,num);
                                        end
                                   else if ((dist <= airng) and (dist <= rh))
                                          and (brg >= ltr) and (brg <= rtr)
                                        then  makecntc (ai,false,ax,ay,nm,num);
                      end;      {    case    }
                      atknext := atknext^.link;
                    end;        {     while   atknext     }
                  end;          {     with     airnext    }
      airnext := airnext^.link;
    end;                {     while  airnext       }
  end;

SEGMENT PROCEDURE SHOWSTATUS;

  begin
    repeat
      screntro(clr);    screntro(off);   writeln;
      writeln('    Please select an option according to which status of  ');
      writeln('    forces report you wish to peruse.    ');
      writeln;
      writeln('          1 :  Ships.                    ');
      writeln('          2 :  Fighter/interceptor.      ');
      writeln('          3 :  Early warning aircraft.   ');
      writeln('          4 :  Radar contacts.           ');
      writeln;
      writeln('          Q :  Quit   ');
      writeln;     writeln;
      writeln('NOTE ::  COORDINATE POSITIONS are SCALED  :   1 = 10 NM.');
      writeln;    writeln;
      writeln('    After your selection you will be presented with    ');
      writeln('    specifics and status information concerning your     ');
      writeln('    selection.  You will then be returned to this menu ');
      writeln('    where you may make another selection or repeat a');
      writeln('    previous selection or quit.   ');
      read (selection);
      while not (selection in ['1'..'4','Q','q']) do
        begin
          screntro(bel);    writeln;
          writeln('     You must select one of the available options. ');
          write  ('              Please try again : ');
          read (selection);    writeln;
        end;    {  while   }
      case selection of
        '1' : shipstatus;
```

```
        '2' : intstatus;
        '3' : aewstatus;
        '4' : rdrstatus;
      end;      {  case  }
    until (selection in ['Q','q']);
  end;

SEGMENT PROCEDURE NEXTEVENTS;

  begin
    screntro(clr);    screntro(off);   writeln;
    writeln('    Please choose your desired course of action : ');
    writeln;
    writeln('          1 :  Move a fighter/interceptor.       ');
    writeln('          2 :  Launch a fighter/interceptor.     ');
    writeln('          3 :  Recover a fighter/interceptor.    ');
    writeln('          4 :  Move an AEW aircraft.             ');
    writeln('          5 :  Launch an AEW aircraft.           ');
    writeln('          6 :  Recover an AEW aircraft.          ');
    writeln('          7 :  Move/manuever an individual ship.');
    writeln('          8 :  Alter the PIM / SOA of the fleet.');
    writeln;
    writeln('          D :  Review the display of forces.     ');
    writeln('          R :  Review the status of forces.      ');
    writeln;
    writeln('          Q :  Quit    ');
    writeln;     writeln;
    writeln('    After your selection you will be asked for specifics ');
    writeln('    about your selection; then you will be returned to this ');
    writeln('    menu where you may make another selection; repeat a ');
    writeln('    selection for another aircraft/ship or quit.   ');
    read (selection);
  end;

SEGMENT PROCEDURE NEXTSTEP;

  begin
    screntro(clr);    screntro(off);    texton;
    write  ('   Current GAME TIME (TOTAL simulated running TIME) is : ');
    writeln(time:3,' minutes. ');     writeln;    writeln;
    if enemies = []
    then begin
           stop := true;
           writeln('  The game will now stop because the attack is dead.');
         end
    else begin
           writeln;     writeln;
           writeln('    Do you wish to stop the program at this time?  ');
           writeln;     yesnosel;     writeln;    writeln;
           if (selection in ['Y','y'])
             then  stop := true
             else
               begin
                 screntro(off);    writeln;   writeln;
```

```
                write  ('   NOTE :: Game WILL STOP after ');
                writeln(endtime:3,' minutes.');
                writeln;     writeln;
                writeln('   Enter desired time step for the next display. ');
                writeln;       writeln;      writeln;
                write  ('   TIME STEP (in minutes) : ');     scrncntro(on);
                len := 2;        tmstep := readint(len);  writeln;   writeln;
                if tmstep > 60
                  then
                    begin
                      writeln;      writeln;
                      writeln('    Confirm your entry of : ',tmstep:3,' minutes.');
                      writeln;       yesnosel;    writeln;    writeln;
                      if selection in ['N','n']
                        then  begin
                                write  ('   Time step (in minutes) : ');
                                tmstep := readint(len);
                                writeln;    writeln;
                              end;
                    end;
              end;
        end;
  end;

SEGMENT PROCEDURE FLTUPDATE;

  begin
    shipnext := shipbase;
    while shipnext <> nil do
      begin
        with shipnext^ do   setnewxy (tmstep,pim,soa,xpos,ypos);
        if shipnext^.class = cv
          then  begin
                  cvx := shipnext^.xpos;
                  cvy := shipnext^.ypos;
                end;
        shipnext := shipnext^.link;
      end;       {    while    }
    airnext := airbase;
    while airnext <> nil do
      begin
        with airnext^ do
          if alt > 0
            then begin
                   if acfrnd = aew
                     then if aewndr <= tmstep
                            then  airdelete (airnext,airbase)
                            else  aewndr := aewndr - tmstep
                     else if intndr <= tmstep
                            then  airdelete (airnext,airbase)
                            else  intndr := intndr - tmstep;
                   if alt = 1
                     then if distance (cvx,cvy,xpos,ypos) <= recov
                            then begin
```

```
                                    xpos    := cvx;
                                    ypos    := cvy;
                                            := 0;
                                    alt     := 0;
                                  end
                              else azmth := degrees (xpos,ypos,cvx,cvy);
                        setnewxy (tmstep,azmth,velcty,xpos,ypos);
                      end
                else begin
                       xpos := cvx;
                       ypos := cvy;
                     end;          {    with / if    }
          airnext := airnext^.link;
        end;        {    while    }
    end;

SEGMENT PROCEDURE ATKUPDATE;

  VAR
    hdg : integer;
    ax,ay : real;

  begin
    atknext := atkbase;
    while atknext <> nil do
      begin
        with atknext^ do   setnewxy (tmstep,azmth,velcty,xpos,ypos);
        atknext := atknext^.link;
      end;        {    while    }
    atknext := atkbase;
    if time >= retrtime
      then   while atknext <> nil do
               begin
                 atknext^.azmth := retrthdg;
                 atknext^.velcty := retrtvel;
                 atknext^.alt := retrtalt;
                 atknext := atknext^.link;
               end
      else if (time div look > 0)
             then begin
                    look := look + incr;
                    while atknext <> nil do
                      begin
                        ax := atknext^.xpos;   ay := atknext^.ypos;
                        if (distance (ax,ay,cvx,cvy) * 10) <= 200
                          then atknext^.alt := inbdalt;
                        atknext^.azmth := degrees (ax,ay,cvx,cvy);
                        atknext := atknext^.link;
                      end;        {    while    }
                  end;
  end;

       {         Beginning text file Thesis3b.       }
```

```
PROCEDURE INITIALIZE;

  begin
    reset (shipdata,'ABA.2:shipdata.data');
    stop := false;      time  := 0;    look := incr;
    frenum := 0;
    shipbase := nil;
    while not eof (shipdata) do
      begin
        frenum := frenum + 1;
        new (shipnext);
        shipnext^ := shipdata^;
        shipnext^.num := frenum;
        if shipnext^.class = cv
          then  begin
                  cvx := shipnext^.xpos;
                  cvy := shipnext^.ypos;
                end;
        shipnext^.link := shipbase;
        shipbase := shipnext;
        get (shipdata);
      end;
    close (shipdata,lock);
  end;

PROCEDURE DOAIRLISTS;

  begin
    reset (airdata ,'ABA.2:airdata.data');
    airbase := nil;         atkbase := nil;  i := 0;
    while not eof (airdata) do
      begin
        case airdata^.iff of
          enemy : begin
                    i := i + 1;
                    new (atknext,enemy);
                    atknext^ := airdata^;
                    atknext^.num := i;
                    atknext^.link := atkbase;
                    atkbase := atknext;
                  end;
          friend : begin
                     frenum := frenum + 1;
                     case airdata^.acfmid of
                       intcpt : begin
                                  new (airnext,friend,intcpt);
                                  airnext^ := airdata^;
                                  airnext^.num := frenum;
                                  airnext^.link := airbase;
                                  airbase := airnext;
                                end;
                       asw    : begin
                                  new (airnext,friend,asw);
                                  airnext^ := airdata^;
```

```
                                    airnext^.num := frcnum;
                                    airnext^.link := airbase;
                                    airbase := airnext;
                                  end;
                      end;          {   case   }
                    end;            {   frnd   }
        end;        {    case    }
        get (airdata);
      end;          {    while   }
    close (airdata,lock);
  end;                        {        DOAIRLISTS      }

PROCEDURE SHOWFLEET  ;

  begin
    grafixmode (bw280,1);    fillport;
    fltnext := fltbase;
    while fltnext <> nil  do
      begin
        nx := round (fltnext^.xpos);
        ny := round (fltnext^.ypos);
        moveto (nx,ny);
        case fltnext^.what of
          boat   :  drawimage (shipform,2,0,0,8,6);
          early  :  drawimage (aewform,2,0,0,10,6);
          fight  :  drawimage (intform,2,0,0,10,6);
          aewcnt :  drawimage (aewrdr ,2,0,0,5,5);
          aicnt  :  drawimage (airdr ,2,0,0,5,5);
          fccnt  :  drawimage (fcrdr ,2,0,0,5,5);
          sscnt  :  drawimage (ssrdr ,2,0,0,5,5);
        end;        {   case    }
        dotat (nx,ny);       str (fltnext^.num,mss);
        moveto (nx,ny+8);    unitwrite(3,mss[1],length(mss),0,12);
        fltnext := fltnext^.link;
      end;
  end;

PROCEDURE SHOWATTACK ;

  begin
    grafixmode (bw280,1);      fillport;
    enenext :- enebase;
    while enenext <> nil  do
      begin
        nx := round (enenext^.xpos);
        ny := round (enenext^.ypos);
        moveto (nx,ny);
        if enenext^.what = plne
          then  drawimage (intform,2,0,0,10,6)
          else  drawimage (ssrdr ,2,0,0,5,5);
        dotat (nx,ny);          str (enenext^.num,mss);
        moveto (nx,ny+8);       unitwrite(3,mss[1],length(mss),0,12);
        enenext := enenext^.link;
      end;
```

```
  end;

PROCEDURE AIRDELETE ;

  VAR
    last : airptr;

  begin
    if thisun = base
      then  base := thisun^.link
      else begin
             last := base;
             while last^.link <> thisun do
               last := last^.link;
             last^.link := thisun^.link;
           end;
  end;

PROCEDURE SHIPDELETE;

  VAR
    last : shiptr;

  begin
    if thisun = base
      then  base := thisun^.link
      else begin
             last := base;
             while last^.link <> thisun do
               last := last^.link;
             last^.link := thisun^.link;
           end;
  end;

PROCEDURE GETKILLS ;

  VAR
    ax,ay,cx,cy   : integer;

  begin
    cntnext := cntbase;
    while cntnext <> nil do
      begin
        atknext := atkbase;
        if cntnext^.dead
          then begin
                 cx := round (cntnext^.xpos);
                 cy := round (cntnext^.ypos);
                 while atknext <> nil do
                   begin
                     ax := round (atknext^.xpos);
                     ay := round (atknext^.ypos);
                     if ((cx = ax)  and (cy = ay))
                       then  airdelete (atknext,atkbase);
```

```
                    atknext := atknext^.link;
                  end;
              end;        { if dead }
          cntnext := cntnext^.link;
        end;        { while cntnext }
      shipnext := shipbase;
      while shipnext <> nil do
        begin
          if shipnext^.sunk then shipdelete(shipnext,shipbase);
          shipnext := shipnext^.link;
        end;          { while shipnext }
    end;

PROCEDURE DISPLAGAME;

    begin
      {$resident whatnext}
      mark (entrtwo);
      positransfer;
      scale := 1;
      atkgraf := false;
      if fltbase <> nil
        then  begin  nucent := fltbase^.num;  recenter;  end;
      repeat
        showfleet ;
        whatnext;
      until selection in ['C','c'];
      if enemlay = []
        then  begin
                texton;  scrncntro(clr);  scrncntro(off);  writeln;  writeln;
                write  ('  The entire attack has been killed, the game will ');
                writeln('halt shortly. ');
                writeln;  writeln;  continue;
              end
        else  begin
                scale := 1;
                atkgraf := true ;
                texton;  scrncntro(clr);  scrncntro(off);  writeln;  writeln;
                writeln('Do you wish to see the attack?');
                writeln;  writeln;  yesnosel;  writeln;
                case selection of
                  'Y','y' : begin
                              if enebase <> nil
                                then  begin
                                        nucent := enebase^.num;
                                        recenter;
                                      end;
                              repeat
                                showattack ;
                                whatnext;
                              until selection in ['C','c'];
                            end;
                end;    { case }
              end;
```

```
    release (entrtwo);
  end;

  PROCEDURE SHIPSTATUS;

  begin
    screentro(clr);   screentro(off);  writeln;
    write  ('   UNIT',' ':17,'X',' ':7,'Y',' ':16,'number   number');
    writeln('  MISSILE  BOMB');
    write  ('  NUMBER    SHIP      POSIT   POSIT   PIM   SOA');
    writeln('  LR-SAM   SR-SAM   HITS    HITS');
    writeln;
    shipnext := shipbase;
    while shipnext <> nil do
      with shipnext^ do
        begin
          case class of
            cv   :  kind := ' carrier ';
            crsr :  kind := ' cruiser ';
            dest :  kind := 'destroyer';
          end;       {   case   }
          write  (num:6,'    ',kind,xpos:8:2,ypos:8:2,pim:6,soa:6);
          writeln(lrsam:7,srsam:9,mhits:7,bhits:8);
          shipnext := link;
        end;      {   with / while  }
    writeln;   writeln;   continue;
  end;

PROCEDURE  INTSTATUS;

  VAR
    deg, dis : integer;

  begin
    screentro(clr);   screentro(off);  writeln;
    write  ('    UNIT       X        Y     relative to');
    writeln(' ':24,'minutes    number');
    write  ('   NUMBER   POSIT   POSIT    CARRIER    HEADING   VELOCITY');
    writeln('   ENDURANCE  MISSILES');
    writeln;
    i := 0;      j := 0;
    airnext := airbase;
    while airnext <> nil do
      with airnext^ do
        begin
          case acfmd of
            intcpt : begin
                       if alt > 0
                         then begin
                                i := i + 1;     j := j + 1;
                                deg := degrees  (cvx,cvy,xpos,ypos);
                                dis := distance (cvx,cvy,xpos,ypos) * 10;
                                write  (num:7,xpos:10:2,ypos:8:2);
                                write  (deg:6,'/',dis:4,'NM');
```

```
                                writeln(azmth:7,velcty:10,intndr:12,eam:10);
                              end
                          else  i := i + 1;
                      end;
          end;       {   case   }
          airnext := link;
        end;       {   with / while  }
    writeln;   writeln;
    writeln('Currently you have, ',(i-j):2,' fighters on board the carrier.');
    writeln;   writeln;   cont

PROCEDURE  AEWSTATUS;

  VAR
    des, dis : integer;

  begin
    screentro(clr);   screentro(off);  writeln;
    write  ('     UNIT       X        Y     relative to');
    writeln(' ':24,'minutes');
    write  ('   NUMBER   POSIT   POSIT    CARRIER');
    writeln('    HEADING   VELOCITY   ENDURANCE');
    writeln;
    i := 0;      j := 0;
    airnext := airbase;
    while airnext <> nil do
      with airnext^ do
        begin
          case acfrnd of
            aew : begin
                    if alt > 0
                      then begin
                             i := i + 1;     j := j + 1;
                             des := degrees  (cvx,cvy,xpos,ypos);
                             dis := distance (cvx,cvy,xpos,ypos) * 10;
                             write  (num:7,xpos:10:2,ypos:8:2);
                             write  (des:6,'/',dis:4,'NM');
                             writeln(azmth:7,velcty:10,intndr:12);
                           end
                      else  i := i + 1;
                  end;
          end;       {   case   }
          airnext := link;
        end;       {   with / while  }
    writeln;   writeln;
    write  ('Currently you have, ',(i-j):2,' early warning aircraft on');
    writeln(' board the carrier.');
    writeln;   writeln;   continue;
  end;
```

```
PROCEDURE  RDRSTATUS;

  VAR
    deg, dis : integer;

  begin
    screentro(clr);   screentro(off);  writeln;
    write  ('    CONTACT      X        Y     relative to   CONTACTING');
    writeln('    UNIT in     DEAD /');
    write  ('    NUMBER     POSIT   POSIT     CARRIER        RADAR    ');
    writeln('    CONTACT     ALIVE');
    writeln;
    i := 0;
    cntnext := cntbase;
    while cntnext <> nil do
      with cntnext^ do
        begin
          deg := degrees  (cvx,cvy,xpos,ypos);
          dis := distance (cvx,cvy,xpos,ypos) * 10;
          case who of
            asrch :  kind := ' air search ';
            ssrch :  kind := 'ship search ';
            fcon  :  kind := 'fire control';
            ai    :  kind := 'interceptor ';
          end;       {   case   }
          if dead
            then  choice := 'killed'
            else  choice := 'alive ';
          i := i + 1;
          write  (num:7,xpos:11:2,ypos:8:2,deg:5,' /',dis:4,'NM   ');
          writeln(kind,rdnm:7,'       ',choice);
          if i mod 15 = 0
            then begin
                   writeln;    writeln;   continue;
                   screentro(clr);   screentro(off);  writeln;
                   write  ('    CONTACT      X        Y     relative to');
                   writeln('    CONTACTING   UNIT in     DEAD /');
                   write  ('    NUMBER     POSIT   POSIT     CARRIER   ');
                   writeln('      RADAR       CONTACT     ALIVE');
                   writeln;
                 end;
          cntnext := link;
        end;       {   with / while  }
    writeln;   writeln;   continue;
  end;

PROCEDURE SELECTOR;

  begin
    while not (selection in ['1'..'8','D','d','R','r','Q','q']) do
      begin
        screentro(bel);     writeln;
        writeln('      You must select one of the available options. ');
        write  ('                Please try again : ');
```

```
        read (selection);    writeln;
      end;    { while }
    case selection of
      '1'      : moveac   (intcpt,'move');
      '2'      : moveac   (intcpt,'launch');
      '3'      : moveac   (intcpt,'recover');
      '4'      : moveac   (aew,'move');
      '5'      : moveac   (aew,'launch');
      '6'      : moveac   (aew,'recover');
      '7'      : moveship ('single');
      '8'      : moveship ('fleet');
      'D','d' : displaysame;
      'R','r' : begin   showstatus;   selection := ' ';   end;
    end;      { case }
  end;

PROCEDURE MAKEOUTFILE;

  begin
    rewrite (shroutfile,'ABA.2:shroutfile.data');
    rewrite (airoutfile,'ABA.2:airoutfile.data');
    shipnext := shipbase;
    while shipnext <> nil do
      begin
        shroutfile^ := shipnext^;
        put (shroutfile);
        shipnext := shipnext^.link;
      end;
    airnext := airbase;
    while airnext <> nil do
      begin
        airoutfile^ := airnext^;
        put (airoutfile);
        airnext := airnext^.link;
      end;
    atknext := atkbase;
    while atknext <> nil do
      begin
        airoutfile^ := atknext^;
        put (airoutfile);
        atknext := atknext^.link;
      end;
    close (shroutfile,lock);
    close (airoutfile,lock);
  end;                          { MAKEOUTFILE }

begin                           { ABAGAME }
  initialize;    doairlists;    showforms;    texton;
  screentro(clr);    screentro(off);    writeln;    writeln;
  writeln('  Do you wish to have the computer step through the game,');
  writeln('  showing only the displays, at a fixed time step?');
  writeln;    writeln;    yesnosel;    writeln;
  case selection of
    'Y','y' : begin
```

```
                writeln;     writeln;
                writeln('   All times are to be entered as MINUTES.');
                writeln;
                write  ('  Enter the length of each step : ');
                len := 2;    tmstep := readint(len);
                writeln;   writeln;  writeln;
                write  ('  Enter the total playing time  : ');
                len := 3;    endtime := readint(len);
                stepthru := true;
              end;
    'N','n' : begin
                tmstep := 0;
                endtime := tmdefault;
                stepthru := false;
              end;
  end;      {   case   }
  repeat
    entbase := nil;
    mark (entrone);              {    MARK and RELEASE are APPLE'S way to  }
    airadarcntr;                 {    build dynamic variables and then     }
    shipradarcntr;               {    release the memory when they are no  }
    displaygame;                 {    longer needed.                       }
    if not stepthru
      then begin
             showstatus;
             repeat
               nextevents;
               selector;
             until selection in ['Q','q'];
             nextstep;
           end;
    time := time + tmstep;
    getkills;
    release (entrone);
    fltupdate;
    atkupdate;
  until ( (time > endtime)   or   stop );
  makeoutfile;
end.                             {      ABAGAME     }
```

LIST OF REFERENCES

1. Low, L., "Theater-Level Gaming and Analysis Workshop for Force Planning, Volume I - Proceedings", Office of Naval Research, 1977.

2. Osborne, A., An Introduction to Microcomputers, Volume 1, Basic Concepts, Osborne/McGraw-Hill, 1980.

3. Waddell, M. C., "Air Battle Analyzer Handbook", Applied Physics Laboratory, 1963.

4. Luerhrmann, A., Packham, H., Apple PASCAL a Hands-On Approach, McGraw-Hill, 1981.

5. Ramsey, H. and Atwood, M., "Human Factors In Computer Systems: A Review of the Literature", Technical Report SAI-79-111-DEN, Englewood, Colorado: Science Applications Inc., September 1979. (NTIS No. ADA -75679)

6. Heyman, E., "FIT - A Federal Income Tax Program in UCSD PASCAL", BYTE, Feb. 82, vol 7, number 2.

7. Anacopa Sciences Inc., "Fundamentals of Human Factors for Engineering and Design - Session 22: Human-Computer Interface Design", Santa Barbara, CA., 1981.

BIBLIOGRAPHY

Apple Computer Inc., APPLE III PASCAL: Programmer's Manual, Volume I, 1981.

Apple Computer Inc., APPLE III PASCAL: Programmer's Manual, Volume II, 1981.

Apple Computer Inc., APPLE III PASCAL: Introduction, Filer, Editor, 1981.

Apple Computer Inc., APPLE III PASCAL: Program Preparation Tools, 1981.

Grogono, P., Programming in PASCAL, Addison-Wesley, 1980.

Zehna, P. (editor), Selected Methods and Models in Military Operations Research, U.S. Government Printing Office, 1971.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Office of Naval Research, Code 230 Attn: Mr. H. Nagelout Arlington, Virginia 22217	1
2.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
3.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
4.	Prof. Alvin P. Andrus, Code 55As Naval Postgraduate School Monterey, California 93940	1
5.	Prof. James D. Esary, Code 55Ey Naval Postgraduate School Monterey, California 93940	1
6.	Lt. James O. Wilson, USN 1117 Birks Lane Virginia Beach, Virginia 23465	3
7.	Prof. Gerald G. Brown, Code 55Bw Naval Postgraduate School Monterey, California 93940	1

```
shipnext := shipbase;
while shipnext <> nil do
  begin
    with shipnext^ do   setnewxy (tmstep,rim,sos,xpos,ypos);
    if shipnext^.class = cv
      then  begin
              cvx := shipnext^.xpos;
              cvy := shipnext^.ypos;
            end;
    shipnext := shipnext^.link;
  end;       {    while    }
airnext := airbase;
while airnext <> nil do
  begin
    with airnext^ do
      if alt > 0
        then begin
               if acfrnd = aew
                 then if aewndr <= tmstep
                        then  airdelete (airnext,airbase)
                        else  aewndr := aewndr - tmstep
                 else if intndr <= tmstep
                        then  airdelete (airnext,airbase)
                        else  intndr := intndr - tmstep;
               if alt = 1
                 then if distance (cvx,cvy,xpos,ypos) <= recov
                        then begin
```


END